

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

IN RE APPLICATION OF: Tatsunori KANAI, et al.

GAU:

SERIAL NO: New Application

EXAMINER:

FILED: Herewith

FOR: METHOD AND SYSTEM FOR PERFORMING REAL-TIME OPERATION USING PROCESSORS

REQUEST FOR PRIORITY

COMMISSIONER FOR PATENTS  
ALEXANDRIA, VIRGINIA 22313

SIR:

- ☐ Full benefit of the filing date of U.S. Application Serial Number \_\_\_\_\_, filed \_\_\_\_\_, is claimed pursuant to the provisions of 35 U.S.C. §120.
- ☐ Full benefit of the filing date(s) of U.S. Provisional Application(s) is claimed pursuant to the provisions of 35 U.S.C. §119(e):  
Application No. \_\_\_\_\_ Date Filed \_\_\_\_\_
- ☒ Applicants claim any right to priority from any earlier filed applications to which they may be entitled pursuant to the provisions of 35 U.S.C. §119, as noted below.

In the matter of the above-identified application for patent, notice is hereby given that the applicants claim as priority:

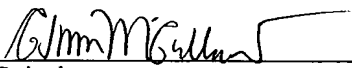
<u>COUNTRY</u>	<u>APPLICATION NUMBER</u>	<u>MONTH/DAY/YEAR</u>
Japan	2003-184975	June 27, 2003

Certified copies of the corresponding Convention Application(s)

- ☒ are submitted herewith
- ☐ will be submitted prior to payment of the Final Fee
- ☐ were filed in prior application Serial No. \_\_\_\_\_ filed \_\_\_\_\_
- ☐ were submitted to the International Bureau in PCT Application Number \_\_\_\_\_  
Receipt of the certified copies by the International Bureau in a timely manner under PCT Rule 17.1(a) has been acknowledged as evidenced by the attached PCT/IB/304.
- ☐ (A) Application Serial No.(s) were filed in prior application Serial No. \_\_\_\_\_ filed \_\_\_\_\_; and
- ☐ (B) Application Serial No.(s) \_\_\_\_\_  
☐ are submitted herewith
- ☐ will be submitted prior to payment of the Final Fee

Respectfully Submitted,

OBLON, SPIVAK, McCLELLAND,  
MAIER & NEUSTADT, P.C.

  
\_\_\_\_\_  
Marvin J. Spivak

Registration No. 24,913

C. Irvin McClelland  
Registration Number 21,124

Customer Number

22850

Tel. (703) 413-3000  
Fax. (703) 413-2220  
(OSMMN 05/03)

0381439

日 本 国 特 許 庁  
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出 願 年 月 日            2 0 0 3 年   6 月 2 7 日  
Date of Application:

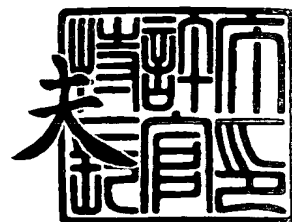
出 願 番 号            特 願 2 0 0 3 - 1 8 4 9 7 5  
Application Number:  
[ST. 10/C] :            [ J P 2 0 0 3 - 1 8 4 9 7 5 ]

出   願   人            株 式 会 社 東 芝  
Applicant(s):

2 0 0 3 年 1 0 月   7 日

特許庁長官  
Commissioner,  
Japan Patent Office

今 井 康



出証番号   出証特 2 0 0 3 - 3 0 8 2 5 9 7

【書類名】 特許願

【整理番号】 A000303288

【提出日】 平成15年 6月27日

【あて先】 特許庁長官 殿

【国際特許分類】 G06F 15/00

【発明の名称】 スケジューリング方法およびリアルタイム処理システム

【請求項の数】 16

【発明者】

    【住所又は居所】 神奈川県川崎市幸区小向東芝町 1 番地 株式会社東芝研  
    究開発センター内

    【氏名】 金井 達徳

【発明者】

    【住所又は居所】 神奈川県川崎市幸区小向東芝町 1 番地 株式会社東芝研  
    究開発センター内

    【氏名】 前田 誠司

【発明者】

    【住所又は居所】 神奈川県川崎市幸区小向東芝町 1 番地 株式会社東芝研  
    究開発センター内

    【氏名】 吉井 謙一郎

【発明者】

    【住所又は居所】 神奈川県川崎市幸区小向東芝町 1 番地 株式会社東芝研  
    究開発センター内

    【氏名】 矢野 浩邦

【特許出願人】

    【識別番号】 000003078

    【氏名又は名称】 株式会社 東芝

## 【代理人】

【識別番号】 100058479

【弁理士】

【氏名又は名称】 鈴江 武彦

【電話番号】 03-3502-3181

## 【選任した代理人】

【識別番号】 100091351

【弁理士】

【氏名又は名称】 河野 哲

## 【選任した代理人】

【識別番号】 100088683

【弁理士】

【氏名又は名称】 中村 誠

## 【選任した代理人】

【識別番号】 100108855

【弁理士】

【氏名又は名称】 蔵田 昌俊

## 【選任した代理人】

【識別番号】 100084618

【弁理士】

【氏名又は名称】 村松 貞男

## 【選任した代理人】

【識別番号】 100092196

【弁理士】

【氏名又は名称】 橋本 良郎

## 【手数料の表示】

【予納台帳番号】 011567

【納付金額】 21,000円

【提出物件の目録】

【物件名】	明細書	1
【物件名】	図面	1
【物件名】	要約書	1
【プルーフの要否】	要	

【書類名】 明細書

【発明の名称】 スケジューリング方法およびリアルタイム処理システム

【特許請求の範囲】

【請求項 1】 複数のプロセッサにリアルタイム処理を実行するためのスレッド群を割り当てるスケジューリング方法であって、

スレッド間の結合属性を示す結合属性情報に基づいて、前記リアルタイム処理を実行するためのスレッド群の中から、互いに協調して動作する複数のスレッドの集合である密結合スレッドグループを選択する選択ステップと、

前記選択ステップによって選択された前記密結合スレッドグループに属するスレッド群がそれぞれ別のプロセッサによって同時に実行されるように、前記密結合スレッドグループに属するスレッド群を当該スレッド群の個数分のプロセッサにそれぞれディスパッチするためのスケジューリング処理を実行するステップとを具備することを特徴とするスケジューリング方法。

【請求項 2】 前記複数のプロセッサはそれぞれローカルメモリを有しており、

別のプロセッサにそれぞれディスパッチされる前記密結合スレッドグループに属するスレッド群の各々の実効アドレス空間の一部に、前記密結合スレッドグループに属する他のスレッドが実行されるプロセッサのローカルメモリをマッピングするステップをさらに具備することを特徴とする請求項 1 記載のスケジューリング方法。

【請求項 3】 前記スケジューリング処理を実行するステップは、前記密結合スレッドグループに属するスレッド群が同一の実行期間に同時に実行されるように、前記密結合スレッドグループに属するスレッド群の個数分のプロセッサそれぞれの実行期間を予約するステップを含むことを特徴とする請求項 1 記載のスケジューリング方法。

【請求項 4】 前記密結合スレッドグループに属するスレッド群の各々は、当該スレッドが実行されるプロセッサのレジスタおよびローカルストレージの内容を示すコンテキスト情報を含むことを特徴とする請求項 1 記載のスケジューリング方法。

【請求項5】 前記複数のプロセッサは共有メモリにそれぞれ電氣的に接続されており、

前記結合属性情報に基づいて、前記リアルタイム処理を実行するためのスレッド群の中から、前記共有メモリ上のバッファを介して通信を行うスレッドの集合である疎結合スレッドグループを選択するステップをさらに具備し、

前記スケジューリング処理を実行するステップは、前記選択された疎結合スレッドグループに属するスレッド群がそれらスレッド間の入出力関係に対応する順序で実行されるように、前記疎結合スレッドグループに属するスレッド群を1以上のプロセッサにディスパッチするためのスケジューリング処理を実行するステップを含むことを特徴とする請求項1記載のスケジューリング方法。

【請求項6】 前記スケジューリング処理は、前記複数のプロセッサの一つによって実行されるオペレーティングシステムによって実行されることを特徴とする請求項1記載のスケジューリング方法。

【請求項7】 ローカルメモリをそれぞれ有する第1および第2のプロセッサにリアルタイム処理を実行するためのスレッド群を割り当てるスケジューリング方法であって、

互いに協調して動作する第1および第2のスレッドがそれぞれ前記第1および第2のプロセッサによって同時に実行されるように、前記第1および第2のスレッドを前記第1および第2のプロセッサにディスパッチするスケジューリング処理を実行するステップと、

前記第2のスレッドが実行される前記第2のプロセッサのローカルストレージを前記第1のプロセッサによって実行される前記第1のスレッドの実効アドレス空間にマッピングするステップとを具備することを特徴とするスケジューリング方法。

【請求項8】 前記第1のスレッドが実行される前記第1のプロセッサのローカルストレージを前記第2のプロセッサによって実行される前記第2のスレッドの実効アドレス空間にマッピングするステップをさらに具備することを特徴とする請求項7記載のスケジューリング方法。

【請求項9】 複数のプロセッサと、

スレッド間の結合属性を示す結合属性情報に基づいて、リアルタイム処理を実行するためのスレッド群の中から、互いに協調して動作する複数のスレッドの集合である密結合スレッドグループを選択する選択手段と、

前記選択手段によって選択された前記密結合スレッドグループに属するスレッド群がそれぞれ別のプロセッサによって同時に実行されるように、前記密結合スレッドグループに属するスレッド群を当該スレッド群の個数分のプロセッサにそれぞれディスパッチするためのスケジューリング処理を実行する手段とを具備することを特徴とするリアルタイム処理システム。

【請求項 10】 前記複数のプロセッサはそれぞれローカルメモリを有しており、

別のプロセッサにそれぞれディスパッチされる前記密結合スレッドグループに属するスレッド群の各々の実効アドレス空間の一部に、前記密結合スレッドグループに属する他のスレッドが実行されるプロセッサのローカルメモリをマッピングする手段をさらに具備することを特徴とする請求項 9 記載のリアルタイム処理システム。

【請求項 11】 前記スケジューリング処理を実行する手段は、前記密結合スレッドグループに属するスレッド群が同一の実行期間に同時に実行されるように、前記密結合スレッドグループに属するスレッド群の個数分のプロセッサそれぞれの実行期間を予約する手段を含むことを特徴とする請求項 9 記載のリアルタイム処理システム。

【請求項 12】 前記複数のプロセッサは共有メモリにそれぞれ電氣的に接続されており、

前記結合属性情報に基づいて、前記リアルタイム処理を実行するためのスレッド群の中から、前記共有メモリ上のバッファを介して通信を行うスレッドの集合である疎結合スレッドグループを選択する手段をさらに具備し、

前記スケジューリング処理を実行する手段は、前記選択された疎結合スレッドグループに属するスレッド群がそれらスレッド間の入出力関係に対応する順序で実行されるように、前記疎結合スレッドグループに属するスレッド群を 1 以上のプロセッサにディスパッチするためのスケジューリング処理を実行する手段を含



むことを特徴とする請求項 9 記載のリアルタイム処理システム。

【請求項 13】 ローカルメモリをそれぞれ有する第 1 および第 2 のプロセッサと、

互いに協調して動作する第 1 および第 2 のスレッドがそれぞれ前記第 1 および第 2 のプロセッサによって同時に実行されるように、前記第 1 および第 2 のスレッドを前記第 1 および第 2 のプロセッサにディスパッチするスケジューリング処理を実行する手段と、

前記第 1 のプロセッサに設けられ、前記第 2 のスレッドが実行される前記第 2 のプロセッサのローカルメモリに対応する物理アドレス空間を、前記第 1 のプロセッサによって実行される前記第 1 のスレッドの実効アドレス空間にマッピングするためのアドレス変換処理を実行するアドレス変換手段とを具備することを特徴とするリアルタイム処理システム。

【請求項 14】 前記第 2 のプロセッサに設けられ、前記第 1 のスレッドが実行される前記第 1 のプロセッサのローカルメモリに対応する物理アドレス空間を、前記第 2 のプロセッサによって実行される前記第 2 のスレッドの実効アドレス空間にマッピングするためのアドレス変換処理を実行するアドレス変換手段をさらに具備することを特徴とする請求項 13 記載のリアルタイム処理システム。

【請求項 15】 複数のプロセッサを含むコンピュータに、リアルタイム処理を実行するためのスレッド群を前記複数のプロセッサに割り当てるスケジューリング処理を実行させるプログラムであって、

スレッド間の結合属性を示す結合属性情報に基づいて、前記リアルタイム処理を実行するためのスレッド群の中から、互いに協調して動作する複数のスレッドの集合である密結合スレッドグループを選択する処理を、前記コンピュータに実行させる手順と、

前記選択された前記密結合スレッドグループに属するスレッド群がそれぞれ別のプロセッサによって同時に実行されるように、前記密結合スレッドグループに属するスレッド群を当該スレッド群の個数分のプロセッサにそれぞれディスパッチするためのスケジューリング処理を、前記コンピュータに実行させる手順とを具備することを特徴とするプログラム。

【請求項 16】 前記複数のプロセッサはそれぞれローカルメモリを有しており、

別のプロセッサにそれぞれディスパッチされる前記密結合スレッドグループに属するスレッド群の各々の実効アドレス空間の一部に、前記密結合スレッドグループに属する他のスレッドが実行されるプロセッサのローカルメモリをマッピングする処理を、前記コンピュータに実行させる手順をさらに具備することを特徴とする請求項 15 記載のプログラム。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は複数のプロセッサにリアルタイム処理を実行するためのスレッド群を割り当てるスケジューリング方法およびリアルタイム処理システムに関する。

【0002】

【従来の技術】

従来より、サーバコンピュータのような計算機システムにおいては、その演算処理能力の向上を図るために、マルチプロセッサ、並列プロセッサのようなシステムアーキテクチャが利用されている。マルチプロセッサおよび並列プロセッサのどちらも、複数のプロセッサユニットを利用することによって演算処理の並列化を実現している。

複数のプロセッサユニットを備えたシステムとしては、例えば、1 台の高速 CPU、複数台の低速 CPU、および共有メモリを備えたシステムが知られている（例えば、特許文献 1 参照）。このシステムにおいては、高速 CPU および複数台の低速 CPU に対する処理プログラムのプロセス群の割付は、プロセス群の並列動作度の大小および処理時間の大小に応じて行われる。

【0003】

また、複数のプロセッサにスレッド群を割り当てるためのスケジューリング技術としては、同一のプロセス内に属するスレッドを同一のプロセッサに実行させる技術が知られている（例えば、特許文献 2 参照）。

【0004】

ところで、最近では、計算機システムのみならず、例えば、A V（オーディオ・ビデオ）データのような大容量のデータをリアルタイムに処理する組み込み機器においても、その演算処理能力の向上のためにマルチプロセッサ、並列プロセッサのようなシステムアーキテクチャの導入が要求されている。

【0005】

【特許文献1】

特開平10-143380号公報

【0006】

【特許文献2】

特開平8-180025号公報

【0007】

【発明が解決しようとする課題】

しかし、マルチプロセッサ、並列プロセッサのようなシステムアーキテクチャを前提としたリアルタイム処理システムの報告はほとんどなされていないのが現状である。

リアルタイム処理システムにおいては、ある許容時間時間の制限内に個々の処理を完了することが要求される。しかし、マルチプロセッサ、並列プロセッサのようなシステムアーキテクチャをリアルタイム処理システムに適用した場合においては、互いに異なるプロセッサユニットによって実行されるスレッド間の通信に関するレイテンシが大きな問題となる。

【0008】

すなわち、通常、互いに異なるプロセッサユニットによって実行されるスレッド間でデータを受け渡すための通信は、共有メモリ上のバッファを介して行う必要がある。これは、それらスレッドそれぞれがどのようなタイミングでどのプロセッサユニットに割り当てられたとしても、データ欠損等の問題が生じないようにするためである。

【0009】

本発明は上述の事情を考慮してなされたものであり、互いに協調して動作するスレッド間のデータの受け渡しを効率よく実行することが可能なスケジューリン

グ方法およびリアルタイム処理システムを提供することを目的とする。

#### 【0010】

##### 【課題を解決するための手段】

上述の課題を解決するため、本発明は、複数のプロセッサにリアルタイム処理を実行するためのスレッド群を割り当てるスケジューリング方法であって、スレッド間の結合属性を示す結合属性情報に基づいて、前記リアルタイム処理を実行するためのスレッド群の中から、互いに協調して動作する複数のスレッドの集合である密結合スレッドグループを選択する選択ステップと、前記選択ステップによって選択された前記密結合スレッドグループに属するスレッド群がそれぞれ別のプロセッサによって同時に実行されるように、前記密結合スレッドグループに属するスレッド群を当該スレッド群の個数分のプロセッサにそれぞれディスパッチするためのスケジューリング処理を実行するステップとを具備することを特徴とする。

#### 【0011】

このように、結合属性情報に基づいて、互いに協調して動作する複数のスレッドの集合である密結合スレッドグループを選択することにより、密結合スレッドグループに属するスレッド群がそれぞれ別のプロセッサによって同時に実行されることを保証することが可能となる。よって、スレッド間の通信は、例えば、互いに相手のスレッドが実行されているプロセッサの例えばレジスタ等を直接アクセスするといった軽量の仕組みによって実現することができ、スレッド間の通信を軽量且つ高速に実行することが可能となる。

#### 【0012】

複数のプロセッサはローカルメモリをそれぞれ有することが好ましい。この場合、密結合スレッドグループに属するスレッド群の各々の実効アドレス空間の一部に、密結合スレッドグループに属する他のスレッドが実行されるプロセッサのローカルストレージをマッピングすることにより、各スレッドが、相手のスレッドが実行されているプロセッサのローカルストレージを直接的にアクセスすることが可能となる。

#### 【0013】

**【発明の実施の形態】**

以下、図面を参照して本発明の実施形態を説明する。

図1には、本発明の一実施形態に係るリアルタイム処理システムを実現するための計算機システムの構成例が示されている。この計算機システムは、リアルタイム性が要求される各種処理をその時間的な制約条件の範囲内で実行する情報処理システムであり、汎用計算機として利用できるほか、リアルタイム性が要求される処理を実行するための様々な電子機器用の埋め込みシステムとして利用することができる。図1に示されているように、この計算機システムにおいては、マスタープロセッサユニット（MPU11：Master Processing Unit）11と、複数のバーサタイルプロセッサユニット（VPU：Versatile Processing Unit）12と、メインメモリ14と、入出力制御装置15とが、接続装置13によって相互に接続されている。接続装置13は、例えば、クロスバススイッチのような相互結合網、あるいはバスによって構成されている。リング状のバス構造を用いることも出来る。MPU11は計算機システムの動作を制御するメインプロセッサである。オペレーティングシステム（OS：Operating System）は、主にMPU11によって実行される。OSの一部の機能はVPU12や入出力制御装置15で分担して実行することもできる。各VPU12は、MPU11の管理の下で各種の処理を実行するプロセッサである。MPU11は、複数のVPU12に処理を振り分けて並列に実行させるための制御を行う。これにより高速で効率よい処理の実行を行うことが出来る。メインメモリ14は、MPU11、複数のVPU12および入出力制御装置15によって共有される記憶装置（共有メモリ）である。OSおよびアプリケーションプログラムはメインメモリ14に格納される。入出力制御装置15には、ひとつあるいは複数の入出力デバイス（入出力装置）16が接続される。入出力制御装置15はブリッジとも呼ばれる。

**【0014】**

接続装置15はデータ転送レートを保証するQoS機能を持つ。この機能は、接続装置15を介したデータ転送を予約されたバンド幅（転送速度）で実行することによって実現される。QoS機能は、たとえば、あるVPU12からメモリ14に5Mbpsでライトデータを送信する場合、あるいはあるVPU12と別のVPU

1 2 との間で 1 0 0 Mbps でデータ転送する場合に利用される。V P U 1 2 は接続装置 1 3 に対してバンド幅（転送速度）を指定（予約）する。接続装置 1 3 は指定されたバンド幅を要求した V P U 1 2 に対して優先的に割り当てる。ある V P U 1 2 のデータ転送に対してバンド幅が予約されたならば、その V P U 1 2 によるデータ転送中に他の V P U 1 2、M P U 1 1 あるいは入出力制御装置 1 5 が大量のデータ転送を行っても、予約されたバンド幅は確保される。この機能は、特に、リアルタイム処理を行う計算機にとって重要な機能である。

#### 【 0 0 1 5 】

図 1 の構成では、M P U 1 1 が 1 つ、V P U 1 2 が 4 つ、メモリ 1 4 が 1 つ、入出力制御装置が 1 つであるが、V P U 1 2 の個数は制限されない。また M P U 1 1 を持たない構成も可能である。この場合、M P U 1 1 の行う処理は、ある一つの V P U 1 2 が担当する。つまり、仮想的な M P U 1 1 の役割を V P U が兼ねる。

#### 【 0 0 1 6 】

図 2 には、M P U 1 1 と各 V P U 1 2 の構成が示されている。M P U 1 1 は処理ユニット 2 1 およびメモリ管理ユニット 2 2 を備えている。処理ユニット 2 1 は、メモリ管理ユニット 2 2 を通してメモリ 1 4 をアクセスする。メモリ管理ユニット 2 2 は、仮想記憶管理と、メモリ管理ユニット 2 2 内のキャッシュメモリの管理を行うユニットである。各 V P U 1 2 は、処理ユニット 3 1、ローカルストレージ（ローカルメモリ） 3 2、およびメモリコントローラ 3 3 を備えている。各 V P U 1 2 の処理ユニット 3 1 は、その V P U 内部のローカルストレージ 3 2 を直接アクセスすることができる。メモリコントローラ 3 3 は、ローカルストレージ 3 2 とメモリ 1 4 の間のデータ転送を行う DMA コントローラの役割を持つ。このメモリコントローラ 3 3 は、接続装置 1 4 の Q o S 機能を利用できるように構成されており、バンド幅を予約する機能および予約したバンド幅でデータ入出力を行う機能を有している。またメモリコントローラ 3 3 は、M P U 1 1 のメモリ管理ユニット 2 2 と同様の仮想記憶管理機能を持つ。V P U 1 2 の処理ユニット 3 1 はローカルストレージ 3 2 を主記憶として使用する。処理ユニット 3 1 はメモリ 1 4 に対して直接的にアクセスするのではなく、メモリコントローラ

33に指示して、メモリ14の内容をローカルストレージ32に転送して読んだり、ローカルストレージ32の内容をメモリ14に書いたりする。

#### 【0017】

MPU11のメモリ管理ユニット22およびVPU12のメモリコントローラ33それぞれによって実行される仮想記憶管理は、たとえば図3のように実施することができる。MPU11の処理ユニット21あるいはVPU12のメモリコントローラ33から見たアドレスは、図3の上の部分に示すような64ビットのアドレスである。この64ビットのアドレスは、上位の36ビットがセグメント番号、中央の16ビットがページ番号、下位の12ビットがページオフセットである。このアドレスから、実際に接続装置13を通してアクセスする実アドレス空間への変換は、セグメントテーブル50およびページテーブル60を用いて実行される。セグメントテーブル50およびページテーブル60は、メモリ管理ユニット22およびメモリコントローラ33に各々設けられている。

#### 【0018】

MPU11および各VPU12から見た実アドレス(RA)空間には、図4に示すように、たとえば以下のようなデータがマッピングされている。

1. メモリ (主記憶装置)
2. MPU11の各種制御レジスタ
3. 各VPU12の各種制御レジスタ
4. 各VPU12のローカルストレージ
5. 各種入出力デバイス (入出力装置) の制御レジスタ (入出力制御装置の制御レジスタも含む)

MPU11および各VPU12は、実アドレス空間の該当するアドレスにアクセスすることで、1～5の各データを読み書きすることができる。特に、実アドレス空間にアクセスすることで、どのMPU11からでも、あるいはどのVPU12からでも、さらに入出力制御装置15からでも、任意のVPU12のローカルストレージ32にアクセスすることができることは重要である。またセグメントテーブルあるいはページテーブルを用いて、VPU12のローカルストレージ32の内容が自由に読み書きされないように保護することもできる。

MPU 11あるいはVPU 12からみたアドレス空間は、図3の仮想記憶メカニズムを用いて、たとえば図5に示すようにマッピングされる。MPU 11あるいはVPU 12上で実行しているプログラムから直接見えるのは、実効アドレス(EA; Effective Address)空間である。EAは、セグメントテーブル50によって、仮想アドレス(VA; Virtual Address)空間にマッピングされる。さらにVAは、ページテーブル60によって、実アドレス(RA; Real Address)空間にマップされる。このRAが、図4で説明したような構造を持っている。

#### 【0019】

MPU 11は制御レジスタ等のハードウェア機構によって、例えば、各VPU 12のレジスタの読み書き、各VPU 12のプログラムの実行開始/停止などの、各VPU 12の管理を行うことができる。また、MPU 11とVPU 12の間、あるいはあるVPU 12と他のVPU 12の間の通信や同期は、メールボックスやイベントフラグなどのハードウェア機構によって行うことができる。

#### 【0020】

この実施形態の計算機システムは、従来ハードウェアで実現されていたようなリアルタイム性の要求の厳しい機器の動作を、ソフトウェアを用いて実現することを可能にする。例えば、あるVPU 12があるハードウェアを構成するある幾つかのハードウェアコンポーネントに対応する演算処理を実行し、それと並行して、他のVPU 12が他の幾つかのハードウェアコンポーネントに対応する演算処理を実行する。

#### 【0021】

図6はデジタルテレビ放送の受信機の簡略化したハードウェア構成を示している。図6においては、受信した放送信号はDEMUX (デマルチプレクサ) 回路101によって音声データと映像データと字幕データそれぞれに対応する圧縮符号化されたデータストリームに分解される。圧縮符号化された音声データストリームはA-DEC (音声デコーダ) 回路102によってデコードされる。圧縮符号化された映像データストリームはV-DEC (映像デコーダ) 回路103によってデコードされる。デコードされた映像データストリームはPROG (プログレッシブ変換) 回路105に送られ、そこでプログレッシブ映像信号に変換する



ためのプログレッシブ変換処理が施される。プログレッシブ変換された映像データストリームはBLEND（画像合成）回路106に送られる。字幕データストリームはTEXT（字幕処理）回路104によって字幕の映像に変換された後、BLEND回路106に送られる。BLEND回路106は、PROG回路105から送られてくる映像と、TEXT回路104から送られてくる字幕映像とを合成して、映像ストリームとして出力する。この一連の処理が、映像のフレームレート（たとえば、1秒間に30コマ、32コマ、または60コマ）に従って、繰り返し実行される。

### 【0022】

図6のようなハードウェアの動作をソフトウェアによって実行するために、本実施形態では、たとえば図7に示すように、図6のハードウェアの動作をソフトウェアとして実現したプログラムモジュール100を用意する。このプログラムモジュール100は、複数の処理要素の組み合わせから構成されるリアルタイム処理を計算機システムに実行させるためのアプリケーションプログラムであり、マルチスレッドプログラミングを用いて記述されている。このプログラムモジュール100は、図6のハードウェアコンポーネント群に対応する複数の処理要素それぞれに対応した手順を記述した複数のプログラム111～116を含んでいる。すなわち、プログラムモジュール100には、DEMUXプログラム111、A-DECプログラム112、V-DECプログラム113、TEXTプログラム114、PROGプログラム115、およびBLENDプログラム116が含まれている。DEMUXプログラム111、A-DECプログラム112、V-DECプログラム113、TEXTプログラム114、PROGプログラム115、およびBLENDプログラム116は、それぞれ図6のDEMUX回路101、A-DEC回路102、V-DEC回路103、TEXT回路104、PROG回路105、およびBLEND回路106に対応する処理を実行するためのプログラムであり、それぞれスレッドとして実行される。つまり、プログラムモジュール100の実行時には、DEMUXプログラム111、A-DECプログラム112、V-DECプログラム113、TEXTプログラム114、PROGプログラム115、およびBLENDプログラム116それぞれに対応する

スレッドが生成され、生成されたスレッドそれぞれが1以上のVPU12にディスパッチされて実行される。VPU12のローカルストレージ32にはそのVPU12にディスパッチされたスレッドに対応するプログラムがロードされ、スレッドはローカルストレージ32上のプログラムを実行する。デジタルテレビ放送の受信機を構成するハードウェアモジュール群それぞれに対応するプログラム111～116と、構成記述117と呼ぶデータとをパッケージ化したものが、デジタルテレビ放送の受信機を実現するプログラムモジュール100になる。

#### 【0023】

構成記述117は、プログラムモジュール100内の各プログラム（スレッド）をどのように組み合わせて実行するべきかを示す情報であり、プログラム111～116間の入出力関係および各プログラムの処理に必要なコスト（時間）などを示す。図8には構成記述117の例が示されている。

#### 【0024】

図8の構成記述117の例では、スレッドとして動作する各モジュール（プログラムモジュール100内の各プログラム）に対して、その入力につながるモジュール、その出力につながるモジュール、そのモジュールの実行に要するコスト、出力につながるモジュールそれぞれへの出力に必要なバッファサイズが記述されている。たとえば、番号③のV-DECプログラムは、番号①のDEMUXプログラムの出力を入力とし、その出力は番号⑤のPROGプログラムに向かっており、その出力に必要なバッファは1MBで、番号③のV-DECプログラム自体の実行コストは50であることを示している。なお、実行に必要なコストは、実行に必要な時間（実行期間）やステップ数などを単位として記述することもできる。また、何らかの仮想的な仕様のVPUで実行した場合の時間を単位とすることも可能である。計算機によってVPUの仕様や処理性能が異なる場合もあるので、このように仮想的な単位を設けてコストを表現するのは望ましい形態である。図8に示した構成記述117に従って実行する場合の、プログラム間のデータの流れは図9の通りである。

#### 【0025】

さらに、構成記述117には、プログラム111～116それぞれに対応する

スレッド間の結合属性を示す結合属性情報がスレッドパラメータとして記述されている。なお、スレッドパラメータはプログラム 1 1 1 ~ 1 1 6 中にコードとして直接記述することも可能である。

#### 【 0 0 2 6 】

次に、図 1 0、図 1 1 を参照して、プログラム 1 1 1 ~ 1 1 6 が本実施形態の計算機システムによってどのように実行されるかを説明する。ここでは、V P U 0 と V P U 1 の 2 つの V P U 1 2 が計算機システムに設けられている構成を想定する。毎秒 3 0 フレームで映像を表示する場合の、各 V P U 1 2 に対するプログラムの割り当てを時間を追って記入したのが図 1 0 である。ここでは周期 1 の間で 1 フレーム分の音声と映像を出力している。まず、V P U 0 で D E M U X プログラムが処理を行い、その結果の音声と映像と字幕のデータをバッファに書き込む。その後 V P U 1 で A - D E C プログラムと T E X T プログラムを順次実行し、それぞれの処理結果をバッファに書き込む。V P U 0 では、次に V - D E C プログラムが映像データの処理を行い、結果をバッファに書き込む。V P U 0 では、続いて P R O G プログラムが処理を行い、結果をバッファに書き込む。この時点で、V P U 1 での T E X T の処理は終わっているので、最後の B L E N D プログラムの実行を V P U 0 で行い、最終的な映像データを作成する。この処理の流れを、毎周期繰り返すように実行する。

#### 【 0 0 2 7 】

ここで説明したように、所望の動作を滞りなく行えるように、各 V P U 1 2 上で、いつ、どのプログラムを実行するかを決める作業を、スケジューリングとよぶ。スケジューリングを行うモジュールをスケジューラとよぶ。本実施形態では、プログラムモジュール 1 0 0 中に含まれる上述の構成記述 1 1 7 に基づいてスケジューリングが行われる。

#### 【 0 0 2 8 】

図 1 1 は、毎秒 6 0 フレームで表示する場合の実行の様子を示している。図 1 0 と異なるのは、図 1 0 では毎秒 3 0 フレームだったので、1 周期 ( 1 / 3 0 秒 ) で 1 フレーム分の処理を完了できたのに対し、図 1 1 では毎秒 6 0 フレーム処理する必要がある点である。すなわち、1 周期 ( 1 / 6 0 秒 ) では 1 フレーム分

の処理を完了できないので、図11では、複数（ここでは2）周期にまたがったソフトウェアパイプライン処理を行っている。たとえば周期1のはじめに入力された信号に対して、VP U 0でDEM U X処理とV-D E C処理を行う。その後、周期2においてVP U 1でA-D E C、T E X T、P R O G、B L E N Dの各処理を行って最終的な映像データを出力する。周期2ではVP U 0は次のフレームのDEM U XとV-D E Cの処理を行っている。このように、VP U 0によるDEM U X、V-D E Cの処理と、VP U 1によるA-D E C、T E X T、P R O G、B L E N Dの処理を、2周期にまたがってパイプライン的に実行する。

#### 【0029】

なお、図7に示したプログラムモジュール100は、本実施形態の計算機システムを組み込んだ機器内のフラッシュROMやハードディスクに予め記録しておいてもよいが、ネットワークを介して流通させるようにしてもよい。この場合、本実施形態の計算機システムによって実行される処理の内容は、ネットワークを介してダウンロードしたプログラムモジュールの種類に応じて決まる。よって、例えば本実施形態の計算機システムを組み込んだ機器に、様々な専用ハードウェアそれぞれに対応するリアルタイム処理を実行させることが出来る。例えば、新しいコンテンツの再生に必要な新しいプレーヤーソフトウェアやデコードソフトウェアや暗号ソフトウェアなどを、本実施形態の計算機システムで実行可能なプログラムモジュールとして、コンテンツと一緒に配布することで、本実施形態の計算機システムを搭載した機器であれば、いずれの機器でも、その能力が許す範囲内で、そのコンテンツを再生することができる。

#### 【0030】

（オペレーティングシステム）

本計算機システムでは、システム内にOS（オペレーティングシステム）をひとつだけ実装する場合には、図12に示すように、そのOS201がすべての実資源（たとえば、MP U 11、VP U 12、メモリ14、入出力制御装置15、入出力装置16など）を管理する。

一方、仮想計算機方式を用いて、複数のOSを同時に動作させることも可能である。この場合には、図13に示すように、まず仮想計算機OS301を実装し

、それがすべての実資源（たとえば、MPU 11、VPU 12、メモリ 14、入出力制御装置 15、入出力装置 16 など）を管理する。仮想計算機 OS 301 はホスト OS と称されることもある。さらに仮想計算機 OS 301 の上に、ひとつ以上の OS（ゲスト OS と呼ぶ）を実装する。各ゲスト OS 302, 303 は、図 14 に示すように、仮想計算機 OS 301 によって与えられる仮想的な計算機資源から構成される計算機上で動作し、ゲスト OS 302, 303 の管理するアプリケーションプログラムに各種のサービスを提供する。図 14 の例では、ゲスト OS 302 は、1 つの MPU 11 と、2 つの VPU 12 と、メモリ 14 とから構成される計算機上で動いていると思っており、ゲスト OS 303 は 1 つの MPU 11 と、4 つの VPU 12 と、メモリ 14 とから構成される計算機上で動いていると思っている。ゲスト OS 302 からみた VPU 12 や、ゲスト OS 303 からみた VPU 12 が、実際には実資源のどの VPU 12 に対応しているかは、仮想計算機 OS 301 が管理している。ゲスト OS 302, 303 は、その対応を意識する必要はない。

### 【0031】

仮想計算機 OS 301 は、計算機システム全体の資源を時分割で各ゲスト OS 302, 303 に割り当てるように、ゲスト OS 302, 303 のスケジューリングを行う。例えば、ゲスト OS 302 がリアルタイム処理を行うものであるとする。たとえば 1 秒間に 30 回、正しいペースで処理を行いたい場合には、各ゲスト OS 302 はそのパラメタを仮想計算機 OS 301 に設定する。仮想計算機 OS 301 は、1/30 秒に 1 回、確実にそのゲスト OS 301 に必要なだけの処理時間が割り当てられるようにスケジューリングを行う。リアルタイム性を要求しない処理を行うゲスト OS には、リアルタイム性を要求するゲスト OS よりも低い優先度で、処理時間の割り当てを行うように、スケジューリングが行われる。図 15 は、時間軸を横にとって、ゲスト OS 302 とゲスト OS 303 が切り替わりながら動いている様子を示している。ゲスト OS 302 が動いている間は、MPU 11 と全ての VPU 12 がゲスト OS 302 の資源として使用され、ゲスト OS 303 が動いている間は、MPU 11 と全ての VPU 12 がゲスト OS 303 の資源として使用される。

**【0032】**

図16は別の動作モードを示している。ターゲットアプリケーションによってはVPU12をずっと占有して利用したい場合がある。たとえば、常にデータやイベントを監視し続けることが必要なアプリケーションがこれに相当する。このようなときには、特定のVPU12を特定のゲストOSによって占有するように、仮想計算機301のスケジューラがスケジュール管理する。図16では、VPU4をゲストOS301の専用資源に指定した場合の例である。仮想計算機OS301がゲストOS302(OS1)とゲストOS303(OS2)を切り替えても、VPU4は常にゲストOS301(OS1)の管理下で動作し続ける。

**【0033】**

さて、複数のVPU12を用いてプログラムを動作させるために、本実施形態では、複数のVPU12それぞれに割り当てるスレッドをスケジューリングするためのスケジューラを含む、VPU実行環境と呼ぶソフトウェアモジュールを用いる。本計算機システムにOSがひとつしか搭載されていない場合は、図17に示すようにそのOS201にVPU実行環境401を実装する。この時、VPU実行環境401は、OS201のカーネル内に実装することもできるし、ユーザプログラムレベルで実装することもできるし、両者に分割して協調して動作するように実装することも出来る。一方、仮想計算機OS上でひとつあるいは複数のOSを動作させる場合、VPU実行環境401を実装する方式には、次のような方式がある。

1. 仮想計算機OS301の中にVPU実行環境401を実装する方式(図18)
2. VPU実行環境401を仮想計算機OS301が管理するひとつのOSとして実装する方式(図19)。図19では、仮想計算機OS301上で動作するゲストOS304自体がVPU実行環境401である。
3. 仮想計算機OS301が管理する各ゲストOSに、それぞれ専用のVPU実行環境401を実装する方式(図20)。図20においては、ゲストOS302, 303にそれぞれVPU実行環境401, 402が実装されている。VPU実行環境401, 402は、仮想計算機OS301の提供するゲストOS間の通

信機能を用いて、必要に応じて、互いに連携して動作する。

4. 仮想計算機OS 301が管理するゲストOSのうちのひとつにVPU実行環境401を実装して、VPU実行環境を持たないゲストOSは、仮想計算機OS 301の提供するゲストOS間の通信機能を用いて、VPU実行環境401を持つゲストOSのVPU実行環境401を利用する方式(図21)。

#### 【0034】

これらの方式のメリットは以下のとおりである。

##### 方式1のメリット

- ・ 仮想計算機OSの持つゲストOS(仮想計算機OSが管理する対象のOS)のスケジューリングと、VPU12のスケジューリングを一体化できるので、効率良く、きめ細かなスケジューリングができ、資源を有効利用できる。

- ・ 複数のゲストOS間でVPU実行環境を共有できるので、新しいゲストOSを導入する場合に新しくVPU実行環境を作らなくてもよい。

##### 方式2のメリット

- ・ 仮想計算機OSの上にあるゲストOS間でVPU12のスケジューラを共有できるので、効率良く、きめ細かなスケジューリングができ、資源を有効利用できる。

- ・ 複数のゲストOS間でVPU実行環境を共有できるので、新しいゲストを導入する場合に新しくVPU実行環境を作らなくてもよい。

- ・ VPU実行環境を仮想計算機OSや特定のゲストOSに依存せずに作れるので、標準化がしやすく、取り替えて使うことも出来る。特定の組み込み機器に適応したVPU実行環境を作って、その機器の特性を活かしたスケジューリング等を行うことで、効率良い実行ができる。

##### 方式3のメリット

- ・ 各ゲストOSに対してVPU実行環境を最適に実装できるので、効率良く、きめ細かなスケジューリングができ、資源を有効利用できる。

#### 【0035】

##### 方式4のメリット

- ・ すべてのゲストOSがVPU実行環境を実装する必要がないので、新しいゲ

スト OS を追加しやすい。

このように、いずれの方式でも V P U 実行環境を実装することができる。また、このほかにも適宜実施可能である。

#### 【 0 0 3 6 】

(サービスプロバイダ)

本実施形態の計算機システムにおいては、V P U 実行環境 4 0 1 は、各 V P U 1 2 に関連する各種資源（各 V P U の処理時間、メモリ、接続装置のバンド幅、など）の管理とスケジューリング機能の他に、さまざまなサービス（ネットワークを使った通信機能、ファイルの入出力機能、コーデックなどのライブラリ機能の呼び出し、ユーザとのインタフェース処理、入出力デバイスを使った入出力処理、日付や時間の読み出し、など）を提供する。これらのサービスは、V P U 1 2 上で動作するアプリケーションプログラムから呼び出されて、簡単なサービスの場合にはその V P U 1 2 上のサービスプログラムで処理される。しかし通信やファイルの処理など V P U 1 2 だけでは処理できないサービスに関しては、M P U 1 1 上のサービスプログラムによって処理する。このようなサービスを提供するプログラムを、サービスプロバイダ（S P）と呼ぶ。

#### 【 0 0 3 7 】

図 2 2 に V P U 実行環境のひとつの実施例を示す。V P U 実行環境の主要部分は M P U 1 1 上に存在する。これが、M P U 側 V P U 実行環境 5 0 1 である。各 V P U 1 2 上には、その V P U 1 2 内で処理可能なサービスを実行する最小限の機能のみを持つ V P U 側 V P U 実行環境 5 0 2 が存在する。M P U 側 V P U 実行環境 5 0 1 の機能は、大きく、V P U コントロール 5 1 1 と、サービスブローカ 5 1 2 の 2 つに分けられる。V P U コントロール 5 1 2 は、主に、各 V P U 1 2 に関連する各種資源（V P U の処理時間、メモリ、仮想空間、接続装置のバンド幅、など）の管理機構や、同期機構や、セキュリティの管理機構や、スケジューリング機能を提供する。スケジューリング結果に基づいて V P U 1 2 上のプログラムのディスパッチを行うのは、この V P U コントロール 5 1 1 である。サービスブローカ 5 1 2 は、V P U 1 2 上のアプリケーションが呼び出したサービス要求を受けて、適当なサービスプログラム（サービスプロバイダ）を呼び出してそ



のサービスを提供する。

VPU側VPU実行環境502は、主に、VPU12上のアプリケーションプログラムが呼び出したサービス要求を受けて、VPU12内で処理できるものは処理し、そうでないものはMPU側VPU実行環境501のサービスブローカ512に処理を依頼する働きをする。

#### 【0038】

図23に、VPU側VPU実行環境502がサービス要求を処理する手順を示す。VPU側VPU実行環境502はアプリケーションプログラムからのサービス呼び出しを受け取ると（ステップS101）、VPU実行環境502内で処理できるサービスであるかどうかを判別し（ステップS102）、それであれば、対応するサービスを実行して、結果を呼び出し元へ返す（ステップS103, S107）。一方、VPU実行環境502内で処理できるサービスではないならば、該当するサービスを実行可能なサービスプログラムがVPU12上で実行可能なプログラムとして登録されているかどうかを判断する（ステップS104）。登録されているならば、当該サービスプログラムを実行し、結果を呼び出し元へ返す（ステップS105, S107）。登録されていないならば、サービスブローカ512に処理を依頼し、そしてサービスブローカ512から返されるサービスの結果を呼び出し元へ返す（ステップS106, S107）。

#### 【0039】

図24に、MPU側VPU実行環境501のサービスブローカ512が、VPU側VPU実行環境502から要求されたサービスを処理する手順を示す。サービスブローカ512はVPU側VPU実行環境502からのサービス呼び出しを受け取ると（ステップS111）、VPU実行環境501内で処理できるサービスであるかどうかを判別し（ステップS112）、それであれば、対応するサービスを実行して、結果を呼び出し元のVPU側VPU実行環境502へ返す（ステップS113, S114）。一方、VPU実行環境501内で処理できるサービスではないならば、該当するサービスを実行可能なサービスプログラムがMPU11上で実行可能なプログラムとして登録されているかどうかを判断する（ステップS114）。登録されているならば、当該サービスプログラムを実行し、

結果を呼び出し元のVPU側VPU実行環境502へ返す（ステップS116，S114）。登録されていないならば、エラーを呼び出し元のVPU側VPU実行環境502へ返す（ステップS117）。

#### 【0040】

なお、VPU12で実行するプログラムが発行するサービス要求には、サービスの実行結果のリプライを返すものもあれば、要求を出すだけでリプライの無いものもある。また、リプライ先は、通常は要求を出したスレッドであるが、リプライ先として他のスレッド、スレッドグループ、あるいはプロセスを指定することもできる。そのため、サービス要求のメッセージには、リプライ先の指定も含めることが好ましい。サービスブローカ512は、広く使われているオブジェクトリクエストブローカを用いて実現することができる。

#### 【0041】

（リアルタイム処理）

本実施形態の計算機システムはリアルタイム処理システムとして機能する。この場合、そのリアルタイム処理システムの対象とする処理は、大きく、

1. ハードリアルタイム処理
2. ソフトリアルタイム処理
3. ベストエフォート処理（ノンリアルタイム処理）

の3種類に分類できる。1と2がいわゆるリアルタイム処理と呼ばれるものである。本実施形態のリアルタイム処理システムは、多くの既存のOSと同様、スレッドとプロセスの概念を持っている。ここではまず、本実施形態のリアルタイム処理システムにおけるスレッドとプロセスに関して説明する。

#### 【0042】

スレッドには、次の3つのクラスがある。

1. ハードリアルタイムクラス

このスレッドクラスは、その時間要件（timing requirements）は非常に重要で、その要件が満たされなかった際に重大な状況になるような、重要なアプリケーションに用いる。

2. ソフトリアルタイムクラス

このスレッドクラスは、例えその時間要件が満たされなかった場合においても、その品質が低下するだけのアプリケーションに用いる。

### 3. ベストエフォートクラス

このスレッドクラスは、その要件の中に一切の時間要件を含まないアプリケーションに用いる。

#### 【0043】

スレッドは本リアルタイム処理システム内において処理を実行する実体である。スレッドには、そのスレッドが実行するプログラムが関連付けられている。各スレッドは、スレッドコンテキストと呼ぶ、それぞれのスレッドに固有の情報を保持している。スレッドコンテキストには、たとえば、プロセッサのレジスタの値や、スタックなどの情報が含まれている。

本リアルタイム処理システムにおいては、MPUスレッドとVPUスレッドの2種類のスレッドが存在する。これら2つのスレッドは、そのスレッドが実行されるプロセッサ（MPU11かVPU12）によって分類されており、スレッドとしてのモデルは全く同じである。VPUスレッドのスレッドコンテキストには、VPU12のローカルストレージ32の内容や、メモリコントローラ33の持つDMAコントローラの状態なども含む。

#### 【0044】

複数のスレッドをグループとしてまとめたものを、スレッドグループと呼ぶ。スレッドグループは、グループに含まれるスレッドすべてに対して同じ属性を与える、などの処理を効率よく簡単にできるメリットがある。ハードリアルタイムクラスまたはソフトリアルタイムクラスのスレッドグループは、密結合スレッドグループ（tightly coupled thread group）と疎結合スレッドグループ（loosely coupled thread group）の2種類に大別される。密結合スレッドグループ（tightly coupled thread group）と疎結合スレッドグループ（loosely coupled thread group）はスレッドグループに付加された属性情報（結合属性情報）によって識別される。アプリケーションプログラム内のコードまたは上述の構成記述によってスレッドグループの結合属性を明示的に指定することができる。

密結合スレッドグループは互いに協調して動作する複数のスレッドの集合から

構成されるスレッドグループである。すなわち、密結合スレッドグループは、そのグループに属するスレッド群が、お互いに密接に連携して動作することを示す。密接な連携とは、例えば、頻繁にスレッド間で通信あるいは同期処理などの相互作用（interaction）を行ったり、あるいは、レイテンシ（latency）（遅延）の小さい相互作用を必要とする場合などである。一方、疎結合スレッドグループは、密結合スレッドグループに比べてそのグループに属するスレッド群間の密接な連携が不要であるスレッドグループであり、スレッド群はメモリ 14 上のバッファを介してデータ受け渡しのための通信を行う。

#### 【0045】

（密結合スレッドグループ）

図 25 に示すように、密結合スレッドグループに属するスレッド群にはそれぞれ別の VPU が割り当てられ、各スレッドが同時に実行される。密結合スレッドグループに属するスレッドを、密結合スレッド（tightly coupled thread）と呼ぶ。この場合、密結合スレッドグループに属する密結合スレッドそれぞれの実行期間がそれら密結合スレッドの個数と同数の VPU それぞれに対して予約され、それら密結合スレッドが同時に実行される。図 25 においては、ある密結合スレッドグループにスレッド A、B の 2 つが密結合スレッドとして含まれており、それらスレッド A、B がそれぞれ VPU 0、VPU 1 によって同時に実行されている様子を示している。スレッド A、B をそれぞれ別の VPU によって同時に実行することを保証することにより、各スレッドは相手のスレッドが実行されている VPU のローカルストレージや制御レジスタを通じて相手のスレッドとの通信を直接的に行うことが出来る。図 26 は、スレッド A、B がそれぞれ実行される VPU 0、VPU 1 のローカルストレージを介してスレッド A、B 間の通信が実行される様子を示している。この場合、スレッド A が実行される VPU 0 においては、そのスレッド A の EA 空間の一部に、通信相手のスレッド B が実行される VPU 1 のローカルストレージ 32 に対応する RA 空間がマッピングされる。このマッピングのためのアドレス変換は、VPU 0 のメモリコントローラ 33 内に設けられたアドレス変換ユニット 331 がセグメントテーブルおよびページテーブルを用いて実行する。スレッド B が実行される VPU 1 においては、そのスレ

ドBのEA空間の一部に、通信相手のスレッドAが実行されるVPU0のローカルストレージ32に対応するRA空間がマッピングされる。このマッピングのためのアドレス変換は、VPU1のメモリコントローラ33内に設けられたアドレス変換ユニット331がセグメントテーブルおよびページテーブルを用いて実行する。図27には、VPU0上で実行されるスレッドAが自身のEA空間にスレッドBが実行されるVPU1のローカルストレージ(LS1)32をマッピングし、VPU1上で実行されるスレッドBが自身のEA空間にスレッドAが実行されるVPU0のローカルストレージ(LS0)32をマッピングした様子が示されている。例えば、スレッドAはスレッドBに引き渡すべきデータがローカルストレージLS0上に準備できた時点で、そのことを示すフラグをローカルストレージLS0またはスレッドBが実行されるVPU1のローカルストレージLS1にセットする。スレッドBはそのフラグのセットに応答して、ローカルストレージLS0上のデータをリードする。

#### 【0046】

このように、結合属性情報によって密結合関係にあるスレッドを特定できるようにすると共に、結合関係にあるスレッドA、Bがそれぞれ別のVPUによって同時に実行されることを保証することにより、スレッドA、B間の通信、同期に関するインタラクションをより軽量で且つ遅延無く行うことが可能となる。

#### 【0047】

(疎結合スレッドグループ)

疎結合スレッドグループに属するスレッド群それぞれの実行時間は、それらスレッド群間の入出力関係によって決定され、たとえ実行順序の制約がないスレッド同士であってもそれらが同時に実行されることは保証されない。疎結合スレッドグループ属するスレッドを、疎結合スレッド(loosely coupled thread)と呼ぶ。図28においては、ある疎結合スレッドグループにスレッドC、Dの2つが疎結合スレッドとして含まれており、それらスレッドC、DがそれぞれVPU0、VPU1によって実行されている様子を示している。図28に示すように、各スレッドの実行時間はばらばらになる。スレッドC、D間の通信は、図29に示すように、メインメモリ14上に用意したバッファを介して行われる。スレッド

CはローカルストレージLS0に用意したデータをDMA転送によってメインメモリ14上に用意したバッファに書き込み、スレッドDはその開始時にDMA転送によってメインメモリ14上のバッファからローカルストレージLS1にデータを読み込む。

### 【0048】

(プロセスとスレッド)

プロセスは、図30に示すように、一つのアドレス空間と一つ以上のスレッドから構成される。一つのプロセスに含まれるスレッドの数と種類は、どのような組み合わせでも構わない。例えば、VPUスレッドのみから構成されるプロセスも構築可能であるし、VPUスレッドとMPUSレッドが混在するプロセスも構築可能である。スレッドがスレッド固有の情報としてスレッドコンテキストを保持しているのと同様に、プロセスもプロセス固有の情報としてプロセスコンテキストを保持する。このプロセスコンテキストには、プロセスに固有であるアドレス空間と、プロセスが含んでいる全スレッドのスレッドコンテキストが含まれる。プロセスのアドレス空間は、プロセスに属するすべてのスレッド間で共有することができる。一つのプロセスは、複数のスレッドグループを含むことができる。しかし、一つのスレッドグループが複数のプロセスに属することはできない。このため、あるプロセスに属するスレッドグループは、そのプロセスに固有であるということになる。本実施形態のリアルタイム処理システムにおいて、スレッドを新しく生成する方式には、Thread first modelとAddress space first modelの2種類がある。Address space first modelは既存のOSで採用されているのと同様の方式で、MPUSレッドにもVPUスレッドにも適用できる。一方、Thread first modelはVPUスレッドにしか適用できない方式で、本発明のリアルタイム処理システムに特有の方式である。Thread first modelでは、既存のスレッド（新しくスレッドを作りたいと思っている側のスレッド。新しく作るスレッドの親になるスレッドのこと。）は、まず新規スレッドが実行するプログラムを指定して、新規スレッドにプログラムの実行を開始させる。この時、プログラムはVPU12のローカルストレージに格納され、所定の実行開始番地から処理が開始される。この時点では、この新規スレッドにはアドレス空間が関連付けられ

ていないので、自身のローカルストレージはアクセスできるが、メモリ 14 はアクセスできない。その後、新規スレッドは、必要に応じて自身で VPU 実行環境のサービスを呼び出してアドレス空間を生成して関連付けたり、MPU 11 側の処理によってアドレス空間を関連付けられたりして、メモリ 14 にアクセスできるようになる。Address space first model では、既存のスレッドは、新しくアドレス空間を生成するか、あるいは既存のアドレス空間を指定して、そのアドレス空間に新規スレッドが実行するプログラムを配置する。そして新規スレッドにそのプログラムの実行を開始させる。Thread first model のメリットは、ローカルストレージだけで動作するので、スレッドの生成やディスパッチや終了処理などのオーバーヘッドを小さくできることである。

#### 【0049】

(スレッド群のスケジューリング)

次に、図 31 のフローチャートを参照して、VPU 実行環境 401 によって実行されるスケジューリング処理について説明する。VPU 実行環境 401 内のスケジューラは、スケジュール対象のスレッド群にスレッドグループ単位で付加されている結合属性情報に基づいて、スレッド間の結合属性をチェックし（ステップ S121）、各スレッドグループ毎にそのスレッドグループが密結合スレッドグループおよび疎結合スレッドグループのいずれであるかを判別する（ステップ S122）。結合属性のチェックは、プログラムコード中のスレッドに関する記述あるいは上述の構成記述 117 中のスレッドパラメータを参照することによって行われる。このようにして、密結合スレッドグループおよび疎結合スレッドグループをそれぞれ特定することにより、スケジュール対象のスレッド群は密結合スレッドグループと疎結合スレッドグループとに分離される。

#### 【0050】

密結合スレッドグループに属するスレッド群に対するスケジューリングは次のように行われる。すなわち、VPU 実行環境 401 内のスケジューラは、スケジュール対象のスレッド群から選択された密結合スレッドグループに属するスレッド群がそれぞれ別の VPU によって同時に実行されるように、その密結合スレッドグループに属するスレッド群と同数の VPU それぞれの実行期間を予約し、ス

レッド群をそれら予約したVPUそれぞれに同時にディスパッチする（ステップS123）。そして、スケジューラは、各スレッドが実行されるVPU内のアドレス変換ユニット331を用いて、各スレッドのEA空間の一部に、協調して相互作用を行う相手となる他のスレッドが実行されるVPUのローカルストレージに対応するRA空間をマッピングする（ステップS124）。一方、スケジューラ対象のスレッド群から選択された疎結合スレッドグループに属する疎結合スレッド群については、スケジューラは、それらスレッド群間の入出力関係に基づいてそれらスレッド群を1以上のVPUに順次ディスパッチする（ステップS125）。

#### 【0051】

（ローカルストレージのマッピング）

本実施形態のリアルタイム処理システムにおいて、MPUSレッドとVPUスレッドの間、あるいはVPUスレッドと他のVPUスレッドの間で、何らかの通信や同期を行いながら協調して動作を行う場合には、協調相手のVPUスレッドのローカルストレージにアクセスする必要がある。たとえば、より軽量で高速な同期機構は、ローカルストレージ上に同期変数を割り付けて実装する。そのため、あるVPU12のローカルストレージを、他のVPU12あるいはMPU11のスレッドが直接アクセスする必要がある。図4に示す例のように、各VPU12のローカルストレージが実アドレス空間に割り付けられている場合、セグメントテーブルやページテーブルを適切に設定すれば、相手のVPU12のローカルストレージを直接アクセスすることができる。しかしこの場合に、大きく2つの問題が生じる。

#### 【0052】

第1の問題は、VPUスレッドのディスパッチ先VPU12の変更に関する問題である。図32のように、VPUスレッドAとBが存在し、それぞれVPU0とVPU1で動いているとする。そして、このスレッドAとBはお互いのスレッドと協調したいので、お互いのスレッドのLS（ローカルストレージ）を、自分のEA空間にマッピングしているとする。また、VPU0, 1, 2のLS0, 1, 2はそれぞれ図32のようにRA空間に存在するとする。この時、VPUスレ



ッドAが、自分のEA空間にマッピングしているのは、VPUスレッドBが動いているVPUのLS、つまり、VPU1のLSであるLS1である。逆に、VPUスレッドBが、自分のEA空間にマッピングしているのは、VPUスレッドAが動いているVPUのLS、つまり、VPU0のLSであるLS0である。その後、VPU実行環境の中のスケジューラによって、VPUスレッドAを実行するVPUがディスパッチされて、VPUスレッドAは、VPU2で動くことになったとする。この時、もはやVPUスレッドAはVPU0では動いていないので、VPUスレッドBが、自分のEA空間にマッピングしているVPU0のLSは、意味がなくなる。この場合、スレッドBが、スレッドAのディスパッチ先VPUが変更になったことを知らなくてもいいように、システムは何らかの方法でLS0にマッピングされていたEA空間のアドレスをLS2にマッピングして、スレッドBから、スレッドAのローカルストレージとしてVPU2のLSであるLS2が見えるようにする必要がある。

#### 【0053】

第2の問題は、物理VPUと論理VPUの対応関係の問題である。VPUをVPUスレッドに割り当てるまでには、実際には、2つのレベルがある。一つは論理VPUのVPUスレッドへの割り当てであり、もう一つが物理VPUの論理VPUへの割り当てである。物理VPUとは、仮想計算機OS301が管理している物理的なVPU12である。そして、論理VPUとは、仮想計算機OS301がゲストOS割り当てた、論理的なVPUのことである。この関係は図14にも示している。たとえば、VPU実行環境401が論理的なVPUを管理する場合、図32の例で、VPUスレッドの割り当て対象となるVPUは論理VPUである。

#### 【0054】

図33は、この2つのレベルの割り当ての概念を示している。直前に説明した第1の問題は、図33の上の段に位置する、VPUスレッドの論理VPUへの割り当て問題に相当する。第2の問題である物理VPUの論理VPUへの割り当て問題は、下の段に位置する割り当てに相当する。図33では、4つの物理VPUから、3つのVPUを選択し、3つの論理VPUに割り当てていることを示して

いる。もし、この物理VPUと論理VPUの対応関係が変わった場合、VPUスレッドの論理VPUへの割り当てが変更になっていなくても、適切な設定の変更が必要となる。例えば、変更後の論理VPUのLSに対するアクセスが、正しい物理VPUのLSを指すように、LSに対応するページテーブルエントリを入れ換える、などである。

#### 【0055】

ある時刻に、図34のように、物理VPU1, 2, 3が論理VPU0, 1, 2にそれぞれ割り当てられているとする。そして、論理VPU1はVPUスレッドAに、そして論理VPU2はVPUスレッドBに割り当てられていたとする。そして、VPUスレッドAとBは、それぞれ、お互いに、相手の動作している物理VPUのLSを自分のEA空間にマッピングしているとする。VPUスレッドAのEA空間にはVPUスレッドBが実行されている物理VPU3のLS3が、そしてVPUスレッドBのEA空間にはVPUスレッドAが実行されている物理VPU2のLS2がマッピングされている。その後、ある時刻に、仮想計算機OS301によって、論理VPU0, 1が物理VPU0, 1に、再割り当てされたとする。すると、今までVPUスレッドAが動作していた論理VPU1は、物理VPU2から物理VPU1へと変化する。論理VPUのVPUスレッドへの割り当ては変化していないが、物理VPUと論理VPUの対応関係が変化したことになる。このため、VPUスレッドBがEA空間にマッピングしている、VPUスレッドAの動作しているVPUのLSを、LS2からLS1に変更し、正しくアクセスできるようにする必要がある。

#### 【0056】

これらの2つの問題を解決するために、本実施形態のリアルタイム処理システムでは、スレッドから見たEA空間の固定アドレスに、必ず相手のスレッドを実行しているVPUのローカルストレージがマップされて見えるように仮想記憶機構を制御する。すなわち、VPUスケジューラによる論理VPUのディスパッチ時、および仮想計算機OS等による物理VPUと論理VPUの対応関係の切り替え時に、適宜ページテーブルやセグメントテーブルを書き換えることで、VPU上で動作しているスレッドからは、いつも同じ番地に相手のスレッドを実行して

いる VPU のローカルストレージが見えるようにする。

### 【0057】

まず、2つのスレッドのEA空間の関係について説明する、2つのスレッドのEA空間は、次の3つのいずれかのパターンで共有あるいは非共有になっている。

1. 共有EA型： 2つのスレッド1, 2がセグメントテーブルもページテーブルも共有している (図35)

2. 共有VA型： 2つのスレッド1, 2は、ページテーブルは共有するが、セグメントテーブルは共有せず、それぞれが持っている (図36)

3. 非共有型： 2つのスレッド1, 2はページテーブルもセグメントテーブルも共有せず、それぞれが持っている (図37)

以下、1の共有EA型を例に、VPUのローカルストレージをどのようにマップするように制御するかについて説明する。

まず、図38に示すように、VA空間上に各論理VPUに対応した領域を設け、そこに、その論理VPUが対応付けられている物理VPUのローカルストレージがマップされるように、ページテーブルを設定する。この例の場合、物理VPU0, 1, 2がそれぞれ論理VPU0, 1, 2に対応付けられている状態を示している。次に、スレッドAからはスレッドBを実行しているVPUのローカルストレージが、固定アドレスであるセグメントaの領域に見えるように、セグメントテーブルを設定する。また、スレッドBからはスレッドAを実行している論理VPUのローカルストレージが、固定アドレスであるセグメントbに見えるように、セグメントテーブルを設定する。この例では、スレッドAは論理VPU2で、スレッドBは論理VPU1で実行している状況を示している。ここで、VPU実行環境401のスケジューラが、スレッドBを論理VPU0にディスパッチしたとする。この時、VPU実行環境401は、図39に示すように、スレッドAからは固定アドレスであるセグメントaを通して、スレッドBを現在実行している論理VPU0のローカルストレージが見えるように、VPU実行環境401はセグメントテーブルを自動的に書き換える。

さらにここで、たとえば仮想計算機OS301がゲストOSのディスパッチを

したため、物理VPUと論理VPUの対応が変化したとする。このとき、たとえば図40に示すように、VPU実行環境401は、ページテーブルを書き換えて、VA空間上に固定されている論理VPUのローカルストレージの領域が、正しい物理VPUのローカルストレージの領域を指すようにする。図40の例では、物理VPU1, 2, 3が論理VPU0, 1, 2に対応するように変更されたため、ページテーブルを書き換えて、現在の正しいマッピングになるようにしている。

#### 【0058】

このように、VPU実行環境401のスケジューラのディスパッチによって、スレッドを実行する論理VPUが変更になった場合には、EA空間からVA空間へのマッピングを行っているセグメントテーブルを書き換えて、第1の問題を解決している。また、仮想計算機OS301などによって、物理VPUと論理VPUの対応が変更になった場合は、VA空間からRA空間へのマッピングを行っているページテーブルを書き換えて、第2の問題を解決している。

このようにして、相互作用を行う相手のスレッドが実行されるプロセッサに応じて、実効アドレス空間にマッピングされる、相手のスレッドに対応するプロセッサのローカルメモリが自動的に変更することにより、各スレッドは相手のスレッドがディスパッチされるプロセッサを意識することなく、相手のスレッドとの相互作用を効率よく行うことが出来る。よって、複数のスレッドを効率よく並列に実行することが可能となる。

#### 【0059】

以上、共有EA型の場合の例を説明したが、2の共有VA型、3の非共有型についても、セグメントテーブルまたはページテーブルを書き換えることにより、同様に第1の問題および第2の問題を解決することができる。

#### 【0060】

上記の第1および第2の問題を解決する別の方法について述べる。ここでも、共有EA型の場合を例に説明する。図41に示すように、協調して動作する複数のVPUスレッドがある場合、それらのスレッドを実行するVPUのローカルストレージを、セグメント上に連続してマップするように、ページテーブルとセグ

メントテーブルを設定する。図41の例の場合、スレッドAは物理VP U 2で、スレッドBは物理VP U 0で実行されており、それぞれのVP Uのローカルストレージが同一のセグメントに連続して配置されるように、ページテーブルとセグメントテーブルを設定している。ここで、VP U実行環境401のスケジューラによってスレッドを実行する論理VP Uがディスパッチされたり、仮想計算機OS 301等によって物理VP Uと論理VP Uの対応が変更になった場合には、それぞれの変更がスレッドAおよびスレッドBに対して隠蔽されるように、ページテーブルを書き換えて、VA空間とRA空間のマップを変更する。たとえば図42は、スレッドAを実行しているVP Uが物理VP U 1に、スレッドBを実行しているVP Uが物理VP U 3に変更になった場合のマッピングを示している。この変更が行われても、スレッドAおよびスレッドBからは、固定したアドレスを持つセグメント内の、所定の領域をアクセスすることで、常に相手のスレッドを実行しているVP Uのローカルストレージをアクセスすることができる。

#### 【0061】

次に、図43のフローチャートを参照して、VP U実行環境401によって実行されるアドレス管理処理の手順について説明する。VP U実行環境401は、各スレッドのEA空間上の固定アドレスに、相手スレッドを実行しているVP Uのローカルストレージに対応するRA空間をマッピングする（ステップS201）。この後、VP U実行環境401は、相手スレッドのディスパッチ先VP Uの変更あるいは論理VP Uと物理VP Uの対応関係の変更に起因して、相手スレッドが実行されるVP Uが変更されたかどうかを判別する（ステップS202）。相手スレッドが実行されるVP Uが変更されたならば、VP U実行環境401は、セグメントテーブルまたはページテーブルの内容を書き換えて、各スレッドのEA空間上の固定アドレスにマッピングされているローカルストレージを、相手スレッドが実行されるVP Uに合わせて変更する（ステップS203）。

#### 【0062】

これまでの例では、蜜結合スレッドグループのように、互いにVP Uによって実行中のスレッド間で、相手のスレッドを実行しているVP Uのローカルストレージをアクセスする方式を説明した。しかし、疎結合スレッドグループなど、協

調して動作するスレッドが必ずしも同時にVPUに割り当てられて実行していない場合も存在する。そのような場合でも、EA空間上には相手のスレッドを実行しているVPU12のローカルストレージをマップする領域は存在するので、その領域を以下のように用いて対処する。

#### 【0063】

第1の方法： 相手のスレッドが実行中で無い場合には、そのスレッドに対応するVPUのローカルストレージをマップする領域にアクセスすると、スレッドは相手のスレッドが実行開始するまで待たされるようにする。

第2の方法： 相手のスレッドが実行中で無い場合には、そのスレッドに対応するVPUのローカルストレージをマップする領域にアクセスすると、スレッドは例外発生やエラーコードによって、その旨を知る。

#### 【0064】

第3の方法： スレッドの終了時に、そのスレッドを最後に実行していたときのローカルストレージの内容をメモリに保存しておき、そのスレッドに対応付けられたローカルストレージを指すページテーブルあるいはセグメントテーブルのエントリからは、そのメモリ領域を指すようにマッピングを制御する。この方式により、相手のスレッドが実行中でなくても、相手のスレッドに対応付けられたローカルストレージがあたかもあるように、スレッドの実行を続けることができる。図44および図45に具体例を示す。

①： いま、スレッドA、BがそれぞれVPU0、1で実行されており、スレッドBのEA空間には相手のスレッドAが実行されているVPU0のローカルストレージLS0がマッピングされているとする。

②： スレッドAの終了時には、スレッドAまたはVPU実行環境401は、スレッドAが実行されているVPU0のローカルストレージLS0の内容をメモリ14に保存する（ステップS211）。

③： VPU実行環境401は、スレッドBのEA空間にマッピングされている相手先スレッドAのローカルストレージのアドレス空間を、VPU0のLS0から、LS0の内容が保存されたメモリ14上のメモリ領域に変更する（ステップS212）。これにより、スレッドBは、相手のスレッドAが実行中でなくな

った後も、その動作を継続することができる。

④： スレッドAに再びVPUが割り当てられたとき、VPU実行環境401は、メモリ14上のメモリ領域をスレッドAが実行されるVPUのローカルストレージに戻す（ステップS213）。たとえばスレッドAに再びVPU0が割り当てられたときは、メモリ14上のメモリ領域の内容は、VPU0のローカルストレージLS0に戻される。

⑤： VPU実行環境401は、スレッドBのEA空間にマッピングされている相手先スレッドAのローカルストレージのアドレス空間を、スレッドAが実行されるVPUのローカルストレージに変更する（ステップS214）。たとえばスレッドAに再びVPU0が割り当てられたときは、スレッドBのEA空間にマッピングされている相手先スレッドAのローカルストレージのアドレス空間は、VPU0のローカルストレージLS0に戻される。

#### 【0065】

なお、スレッドAにVPU2が割り当てられたときは、メモリ14上のメモリ領域の内容は、VPU2のローカルストレージLS2に復元される。そして、スレッドBのEA空間にマッピングされている相手先スレッドAのローカルストレージのアドレス空間は、VPU2のローカルストレージLS2に変更される。

#### 【0066】

（スレッドの状態遷移）

一般にスレッドは、生成されてから消滅するまで、たとえば図46に示すような状態遷移を行う。図46の例では、以下の7種類の状態を遷移する。

##### 1. NOT EXISTENT状態

論理的な状態であり、有効なスレッドでは、この状態はない。

##### 2. DORMANT状態

スレッドは生成されているが、まだ実行は開始されていない。

##### 3. READY状態

スレッドが、その実行を開始する準備ができている状態。

##### 4. WAITING状態

スレッドが、実行を開始（再開）するための条件が満たされることを待っている

る状態。

#### 5. RUNNING状態

スレッドが、実際にVPUまたはMPU上で実行されている状態。

#### 6. SUSPENDED状態

VPU実行環境や他のスレッドにより、スレッドが強制的にその実行を中断させられている状態。

#### 7. WAITING-SUSPENDED状態

WAITING状態とSUSPENDED状態が重なった状態。

### 【0067】

これらの7つの状態の間の遷移条件と、その遷移に伴うスレッドコンテキストの扱いは、以下ようになる。

#### <NOT EXISTENT状態からDORMANT状態への遷移>

- ・スレッドの作成によって遷移する。
- ・スレッドコンテキストが作成される。ただしコンテキストの中身は初期状態である。

#### <DORMANT状態からNOT EXISTENT状態への遷移>

- ・スレッドの削除によって遷移する。
- ・スレッドが、そのスレッドコンテキストを保存するように設定されていた場合、この遷移によって、保存されていたコンテキストは破棄される。

#### <DORMANT状態からWAITING状態への遷移>

- ・スレッドが、実行環境に対してスレッドのスケジューリングをリクエストすると、スレッドの状態は、DORMANT状態からWAITING状態へ遷移する。

#### <WAITING状態からREADY状態への遷移>

- ・スレッドが、生起するのを待っていたイベント（例えば、同期や通信、タイマなど）が、生起した場合に、スレッドの状態はWAITING状態からREADY状態へ遷移する。

### 【0068】

#### <READY状態からRUNNING状態への遷移>

- ・スレッドが、実行環境によってMPUまたはVPUにディスパッチされると



、スレッドの状態は、READY状態からRUNNING状態へ遷移する。

#### 【0069】

・スレッドコンテキストがロードされる。また、スレッドコンテキストが退避されていた場合には、復元される。

<RUNNING状態からREADY状態への遷移>

・スレッドが、スレッドの実行を横取りされると、スレッドの状態は、RUNNING状態からREADY状態へ遷移する。

#### 【0070】

<RUNNING状態からWAITING状態への遷移>

・スレッドが、同期や通信などの機構を利用し、イベントを待つために自身の実行を中断した場合、スレッドの状態は、RUNNING状態からWAITING状態へ遷移する。

・すべてのクラスのスレッドは、スレッドコンテキストを保存するように設定することができる。スレッドが、スレッドコンテキストを保存するように設定されていた場合は、RUNNING状態からWAITING状態へ遷移する際に、実行環境によって、そのスレッドのスレッドコンテキストが退避される。このスレッドコンテキストは、DORMANT状態に遷移しない限り保持され、次にこのスレッドがRUNNING状態に遷移した時に復元される。

<RUNNING状態からSUSPENDED状態への遷移>

・スレッドが、実行環境や他のスレッドからの指示などによって、強制的にその実行を中断させられた場合、スレッドの状態は、RUNNING状態からSUSPENDED状態へ遷移する。

・すべてのクラスのスレッドは、スレッドコンテキストを保存するように設定することができる。スレッドが、スレッドコンテキストを保存するように設定されていた場合は、RUNNING状態からSUSPENDED状態へ遷移する際に、実行環境によって、スレッドコンテキストが退避される。このスレッドコンテキストは、DORMANT状態に遷移しない限り、次にこのスレッドがRUNNING状態に遷移した時に復元される。

#### 【0071】

<RUNNING状態からDORMANT状態への遷移>

・スレッドは、スレッド自身でその実行を終了した場合に、RUNNING状態からDORMANT状態へ遷移する。

・スレッドが、そのスレッドコンテキストを保存するように設定されていた場合、この遷移によってコンテキストの内容が破棄される。

<WAITING状態からWAITING-SUSPENDED状態への遷移>

・スレッドがWAITING状態にてイベントなどの生起を待っている最中に、外部から強制的にスレッドの実行を中断された場合、スレッドの状態は、WAITING状態からWAITING-SUSPENDED状態へ遷移する。

<WAITING-SUSPENDED状態からWAITING状態への遷移>

・スレッドが、WAITING-SUSPENDED状態にいる最中に、外部からスレッドの実行を再開された場合、スレッドの状態はWAITING-SUSPENDED状態からWAITING状態へ遷移する。

【 0 0 7 2 】

<WAITING-SUSPENDED状態からSUSPENDED状態への遷移>

・スレッドは、スレッドがWAITING状態にいた時に待っていたイベントが生起した場合に、スレッドの状態は、WAITING-SUSPENDED状態からSUSPENDED状態へ遷移する。

【 0 0 7 3 】

<SUSPENDED状態からREADY状態への遷移>

・スレッドが、外部からスレッドの実行を再開させられた時に、スレッドの状態は、SUSPENDED状態からREADY状態へ遷移する。

【 0 0 7 4 】

<READY状態からSUSPENDED状態への遷移>

・スレッドが、外部環境によってスレッドの実行が中断させられた場合に、スレッドの状態は、READY状態からSUSPENDED状態へ遷移する。

【 0 0 7 5 】

(スレッドの実行期間)

スレッドに実際にV P Uが割り当てられて処理を実行しているRUNNING状態の

期間を、実行期間 (execution term) と呼ぶ。一般にスレッドが生成されてから消滅するまでの間には、スレッドは複数の実行期間を持つ。図 47 はあるスレッドの生成から消滅までの時間軸に沿った状態の変化の例を示しているが、この例では、その生存期間中に、2 回の実行期間があることを示している。実行期間と実行期間との間のコンテキストの保存 (save) や復元 (restore) は、さまざまな方法を用いて実現することができる。たとえば、多くの通常のスレッドは、実行期間が終了した時点のコンテキストを保存しておいて、次の実行期間の初めにそのコンテキストを復元するように動作させる。一方、ある種の周期的な (periodic) 処理においては、全ての周期 (period) において、実行期間の開始時には新しいコンテキストを作成してその実行期間中はそのコンテキストを使って実行を進め、実行期間の終了時にはそのコンテキストは廃棄するように動作させる。

#### 【0076】

(蜜結合スレッドグループに属するスレッドの実行期間)

蜜結合スレッドグループに属するスレッドの場合の実行期間は、たとえば図 48 のようになる。すなわち、蜜結合スレッドグループに属するすべてのスレッドは、ある一つの実行期間において、全てのスレッドが同時に実行されるように、VPU 実行環境 401 によってスケジューリングされる。このような蜜結合スレッドグループは、主としてハードリアルタイムスレッドに対して使用される。そのため、この動作を実現するために、VPU 実行環境 401 は、ハードリアルタイムクラスにおける実行期間を予約するときに、同時に使用するプロセッサとその数を指定する。さらに、VPU 実行環境 401 は、予約するそれぞれのプロセッサに対して、一対一に同時実行させるスレッドのコンテキストを対応させる。

#### 【0077】

なお、ある期間において蜜結合スレッドグループに属していた複数のスレッドは、他の実行期間においては、蜜結合の関係を解消して、各スレッドが別々に動作することもできる。このような場合には、各スレッドは、今、蜜結合スレッドとして動作しているのか、あるいは、別々に動作しているのかを意識して、相手のスレッドとの通信や同期等の処理を行う必要がある。各スレッドには、横取り可能 (preemptive) か横取り不可 (non-preemptive) を示すプリエンプションに

関連した属性が与えられる。Preemptiveとは、スレッドの実行期間中に、そのスレッドが横取りされることを許す、すなわち、実行を停止させることができるという属性である。Non-preemptiveとは、スレッドの実行期間中に、そのスレッドが横取りされないことを保障するという属性である。この横取り不可 (non-preemptive) という属性の意味は、スレッドのクラス間で異なる。ハードリアルタイムクラスでは、スレッドが実行を開始すると、実行期間が終わるまで、そのスレッド自身以外、誰もその実行を止めることが出来ないことを意味する。ソフトリアルタイムクラスでは、そのクラスにとって、横取り可能性 (preemptive-ness) は必要不可欠であるため、横取り不可の属性はサポートされない。ベストエフォートクラスでは、スレッドの実行は、他のベストエフォートクラスからの横取りからは保護されるものの、ハードリアルタイムやソフトリアルタイムクラスといったより高いレベルからは、横取りされる。

#### 【0 0 7 8】

(スレッドの実行モデル)

スレッドの実行モデルは、大きく、図 4 9 に示すような周期実行モデルと、図 5 0 に示すような非周期実行モデルの 2 つに分類できる。周期実行モデルでは、スレッドは周期的 (periodically) に実行される。その一方、非周期実行モデルでは、イベントを起点としてその実行が行われる。周期実行モデルの実装方式には、ソフトウェア割込みを用いる方式と、同期機構 (synchronization primitives) のようなイベントオブジェクトを用いる方式がある。ハードリアルタイムクラスでは、ソフトウェア割込みを用いて実装する。すなわち、V P U 実行環境 4 0 1 は、周期的な処理を開始するタイミングで、所定の方法で決定されるスレッドのエントリポイントへジャンプしたり、あるいは、事前に所定の手順で登録されたコールバック関数を呼び出す。ソフトリアルタイムクラスでは、イベントオブジェクトを用いて実装する。すなわち、各周期において、あらかじめ登録されたイベントオブジェクトに対して、たとえば V P U 実行環境 4 0 1 がイベントを通知するので、ソフトリアルタイムスレッドは、毎周期そのイベントオブジェクトを待ち、イベントが発生したら所定の処理を実行するようにプログラムを構成することで、周期実行モデルを実現する。ベストエフォートクラスの場合は、ソ

フトウェア割込みを用いる方式を用いても、イベントオブジェクトを用いる方式を用いても、周期実行モデルを実装できる。なお、実際の実行は、それぞれの周期の先頭で常に開始されるとは限らず、制約条件 (constraints) の範囲内で、状態に応じて遅らされることもある。

#### 【0079】

非周期実行モデルは、イベントモデルを用いると、周期実行モデルと同様に実現できる。すなわち、ソフトリアルタイムやベストエフォートクラスでは、非周期実行モデルは、イベントが通知されるタイミングが異なるだけで、実装手法上、は周期実行モデルと同じになる。ハードリアルタイムクラスの場合は、時間要件を保障するために必要な、最小発生期間 (minimum inter-arrival time) やデッドラインは、システムの振る舞いを強く制約するため、非周期実行は制限される。

#### 【0080】

(コンテキストの切り替え)

本実施形態のリアルタイム処理システムにおいては、VPUスレッドの実行期間の終了に伴うコンテキストの切り替え方式は、複数の方式から選択することができる。VPUのコンテキスト切り替えのコストは非常に大きいので、その方式を選択できるようにすることで、コンテキスト切り替えの効率を向上させることができる。指定したコンテキスト切り替え方式は、スレッドの予約された実行期間が終了した際に用いられるものである。実行期間中のコンテキスト切り替え、すなわち、いわゆるプリエンプションの際には、どの様な場合においても現在のスレッドの全てのコンテキストを保存して、次に実行再開するときに復元する必要がある。本実施形態のリアルタイム処理システムで提供するVPUコンテキスト切り替えの方式には、たとえば、以下のような方式がある。

#### 【0081】

##### 1. コンテキストの破棄

いかなるコンテキストも保存しない。

##### 2. 完全なコンテキストの保存

VPUのレジスタ、ローカルストレージ、およびメモリコントローラ内のDM

Aコントローラの状態を含む、VPUの完全なコンテキストを保存する。

### 3. Graceful コンテキスト保存

VPUのメモリコントローラ内のDMAコントローラが実行中の全ての動作が完了するまでコンテキスト切り替えを遅延する。その後、VPUのレジスタとローカルストレージの内容を保存する。この方式では、完全なコンテキスト保存と同様、VPUのコンテキストの全てが保存される。

#### 【0082】

スレッドのスケジューリングを行うスケジューラは、MPUSレッドとVPUスレッドの両方をスケジューリングするひとつのスケジューラとして実装することもできるし、MPUSレッド用のスケジューラとVPUスレッド用のスケジューラを別に実装することもできる。MPUとVPUではコンテキスト切り替えのコストが異なるため、別々にそれぞれに適したスケジューラを実装するほうが効率よくなる。

#### 【0083】

(ハードリアルタイムクラスのスケジューリング)

ハードリアルタイムクラスのスレッド群のスケジューリングは、タスクグラフを拡張した予約グラフを用いて行われる。図51はタスクグラフの例である。タスクグラフは、タスク間の関係を表す。タスク間の矢印は、タスク間の依存関係（入出力関係）を示している。図51の例では、タスク1とタスク2は、自由に実行を開始することが出来ることを表している。それに対し、タスク3は、タスク1とタスク2両方の実行終了後に始めて実行を開始することが出来ることを表している。また、タスク4とタスク5は、タスク3の実行終了後に実行を開始することが出来ることを表している。タスクグラフにはコンテキストの概念がない。例えば、タスク1とタスク4とを同じコンテキストを用いて実行したい場合に、それを記述することができない。そこで、本実施形態のリアルタイム処理システムでは、以下のようにしてタスクグラフを拡張した予約グラフを用いる。

#### 【0084】

まず、タスクグラフを、タスクではなく実行期間の間の関係を示すものととらえる。そして、それぞれの実行期間に、コンテキストを関係付けることで、その

コンテキストに対応するスレッドが、その実行期間に実行されることを示す。複数の実行期間に同じコンテキストが関係付けると、それら全ての実行期間において、そのスレッドが実行されることを示す。例えば、図52では、スレッド1のコンテキストが実行期間1と2とに関係付けられており、スレッド1は、実行期間1と2の期間で実行されることを示す。さらに、グラフに用いられる実行期間の間の矢印に、実行環境にて保障されるハードリアルタイムの制約条件を表す属性を付加する。このようにして作成した予約グラフを用いて、リアルタイムシステムアプリケーションのモデルを一切修正することなく、処理モデルとその処理が持つ時間要件などの制約条件を記述することが可能になる。図53に、図52をベースに作成した予約グラフの例を示す。図53でコンテキスト1, 2, 3は、それぞれ図52のスレッド1, 2, 3のコンテキストを示している。

#### 【0085】

(ソフトリアルタイムクラスのスケジューリング)

ソフトリアルタイムクラスのスケジューリングは、スレッドの実行形態を予測可能とするために、固定優先度スケジューリングを用いて実行される。そのスケジューリング方式としては、固定優先度FIFOスケジューリングと固定優先度ラウンドロビンスケジューリングの2種類のスケジューリングアルゴリズムを用意する。優先度の高いスレッドの実行を優先するため、低い優先度のスレッドが実行中であっても、より高い優先度のスレッドが実行可能になった場合には、低優先度のスレッドの実行をプリエンプトし、直ちに高優先度のスレッドの実行を開始する。クリティカルセクション(critical section)の実行時に発生する、優先度逆転問題を避けるため、優先度継承プロトコルや、優先度シーリングプロトコルなどの同期機構を併せて実施するのが望ましい。

#### 【0086】

(ベストエフォートクラスのスケジューリング)

ベストエフォートクラスのスケジューリングは、たとえば、動的優先度スケジューリングなどを用いる。

#### 【0087】

(階層型スケジューラ)

VPU実行環境401内のスケジューリング機能は、図54に示すような階層型のスケジューラとして実施することができる。すなわち、スレッドレベルのスケジューリングは、スレッドクラス間（inter-class）スケジューリングと、スレッドクラス内（intra-class）スケジューリングの、2つの階層により構成する。そのため、VPU実行環境401内のスケジューラは、スレッドクラス内（intra-class）スケジューリング部601と、スレッドクラス間（inter-class）スケジューリング部602とを持つ。スレッドクラス間スケジューリングでは、スレッドクラス間を跨るスケジューリングを行う。スレッドクラス内スケジューリングでは、それぞれのスケジューリングクラスごとに、そのスケジューリングクラスに属するスレッドのスケジューリングを行う。スレッドクラス内（intra-class）スケジューリング部601には、ハードリアルタイム（ハードRT）クラススケジューリング部611、ソフトリアルタイム（ソフトRT）クラススケジューリング部612、ベストエフォートクラススケジューリング部613が設けられている。

#### 【0088】

スレッドクラス間スケジューリングとスレッドクラス内スケジューリングは、階層構造をなしており、まず、スレッドクラス間スケジューリングが動作して、どのスレッドクラスを実行するか決定した後、該当するスレッドクラス内スケジューリングによって、そのスレッドクラス内のどのスレッドを実行するかを決定する。スレッドクラス間スケジューリングは、プリエンプト可能な固定優先度スケジューリングを用いる。このとき、ハードリアルタイムクラスが最高優先度を持ち、ソフトリアルタイムクラス、ベストエフォートクラスの順に優先度が低くなるようにする。低優先度クラスのスレッドは、より優先度の高いクラスのスレッドが実行可能（ready）になると、その実行はプリエンプトされる。スレッドクラス間の同期は、VPU実行環境401によって提供される同期プリミティブによって実現する。このとき特に、ハードリアルタイムスレッドにはブロックすることのないプリミティブのみ使用できるようにして、ハードリアルタイムスレッドのブロックが発生しないようにする。また、ベストエフォートスレッドがソフトリアルタイムスレッドをブロックした場合には、そのベストエフォート



スレッドは、ソフトリアルタイムスレッドとして扱うことで、スレッドクラス間の優先度逆転の発生を防止するようにする。さらに、そのベストエフォートスレッドが、他のソフトリアルタイムスレッドによってブロックされるような場合には、優先度継承プロトコルなどの方式を用いてブロックされないようにする。

#### 【0089】

(スレッドパラメタ)

本実施形態のリアルタイム処理システムでは、さまざまなパラメタを用いてスケジューリングを行う。各クラスのスレッドに共通のパラメタには、たとえば以下のようなものがある。

- ・スレッドのクラス（ハードリアルタイム、ソフトリアルタイム、ベストエフォート）
- ・使用するリソース（MPUもしくはVPUの数、バンド幅、物理メモリサイズ、入出力デバイス）
- ・優先度
- ・横取り可能（preemptive）か横取り不可（non-preemptive）か

さらにハードリアルタイムクラスのスレッドに関しては、たとえば以下のようなパラメタがある。

#### 【0090】

- ・実行期間
- ・デッドライン
- ・周期あるいは最小発生期間（minimum inter-arrival time）
- ・VPUのコンテキスト切り替え方式

図55にハードリアルタイムクラスの基本的なパラメタの例を示す。図57の一番上にある例1の実行期間の予約指定の例では、指定した実行期間の間、MPUを1つ、VPUを2つ同時に予約し、VPUのコンテキストを完全に保存することを指定している。この場合、3つのプロセッサ上で同時にスレッドが実行され、その実行期間終了後に、MPUスレッドに加え、VPUスレッドのコンテキストが完全に保存される。次に、右上にある例2では、VPU数とその実行期間によって表現される処理が、デッドラインより以前に実行されることを保障する

際の、デッドラインの指定方法を示している。デッドラインは、予約リクエストを行った時刻(request time)からの相対時刻で指定される。もっとも下にある例 3 では、周期実行を指定している。この例では、2つの VPU 12 を指定した実行期間が、周期的に実行され、また、各周期の実行後に VPU スレッドのコンテキストが破棄され、全ての処理が新しいコンテキストで処理されること示している。さらに、その周期の先頭からの相対時刻を用いてデッドラインを指定している。

#### 【0091】

ハードリアルタイムクラスで用いる別のパラメタとして、たとえば以下に示すような制約条件がある。

- ・ タイミング制約 (絶対タイミング制約、相対タイミング制約)
- ・ 先行制約
- ・ 相互排他制約

タイミング制約は、実行タイミングを遅らせる手段を提供する。絶対タイミング制約は、図 56 に示すように、例えば周期の開始時刻のような、ある静的なタイミングを基準として遅延時間を指定する制約条件である。相対タイミング制約は、図 57 に示すように、例えば他の実行期間の開始時刻や終了時刻のような、動的なタイミングやイベントを基準として許容可能な遅延時間を指定する制約条件である。先行制約は、相対タイミング制約を用いて、他の実行期間の終了時間を基準にし、その遅延時間を 0 以上と指定することで実現できるので、先行制約は相対タイミング制約の特殊な場合と考えることができる。

#### 【0092】

相互排他制約 (mutual exclusive) は、図 58 に示すように、それぞれの実行期間が、時間的に重ならないことを保障する制約である。相互排他制約を用いることによって、ロックによって発生するスレッド実行時間の予測不可能性を削減することが可能になる。すなわち、あるリソースを共有する全てのスレッドが同時に実行されないようにして、そのリソースに関するロックをなくすことができる。

#### 【0093】

(スレッドの同期機構)

本実施形態のリアルタイム処理システムでは、スレッドの同期機構として、たとえば以下のような手段を用いる。

【0094】

- ・セマフォ
- ・メッセージキュー
- ・メッセージバッファ
- ・イベントフラグ
- ・バリア
- ・ミューテックス

その他の同期プリミティブも、これらと同様に用いることができる。このような同期機構を実現する手段として、本発明のリアルタイム処理システムでは、次の3通りの方式がある。

【0095】

- ・メモリ（主記憶）13あるいはVPUのローカルストレージ32上に、たとえばTEST&SETのような命令を使って実現する
- ・メールボックスやシグナルレジスタなどのハードウェア機構を使って実現する
- ・VPU実行環境がサービスとして提供する機構を利用する

これらの実現手段の異なる同期機構は、それぞれ得失を持っているため、それを利用するスレッドの属性等によって、たとえば図59のように使い分けるのが望ましい。すなわち、MPUやVPUが共有してアクセスできるメモリ13（主記憶MS）を使って実装した同期機構は、すべてのクラスのスレッドで使用できる。それに対して、VPU12のローカルストレージLS上に実装した同期機構は、密結合スレッドグループ（tightly coupled thread group）に属するスレッドのみが使うことができる。これは、密結合スレッドグループに属するスレッドのみが、同期相手のスレッドが同時に動作していることを保障されるからである。例えば、相手のスレッドが動作しているVPUのローカルストレージ上に実装した同期機構を用いる場合、密結合スレッドグループのスレッドであれば、同期

機構を使う時点で、相手のスレッドが動作していることが保障されているので、その相手スレッドを実行しているVPUのローカルストレージに同期機構のための情報が必ず存在する。

#### 【0096】

メモリ（主記憶MS）やローカルストレージLS以外の手段を用いて実装した同期機構としては、ハードウェア機構を使って実現する場合と、VPU実行環境401のサービスを使う場合がある。密結合スレッドグループに属するスレッド、あるいはハードリアルタイムクラスのスレッドは、速い同期機構が必要であるので、ハードウェア機構を用いて実装した同期機構を用いるのが望ましい。それに対して、疎結合スレッドグループに属するスレッド、あるいはソフトリアルタイムクラスと、ベストエフォートクラスのスレッドは、実行環境が提供する機構を利用するのが望ましい。

#### 【0097】

##### （同期機構の自動選択）

本実施形態のリアルタイム処理システムでは、上記の同期機構を、スレッドの属性や状態に合わせて自動的に選択・切り替えを行うことができる。これは例えば図60に示すような手順により、同期処理を行いたいスレッドが密結合スレッドグループに属している状態の間は（ステップS201のYES）、メモリ14あるいはVPU12のローカルストレージ32あるいはハードウェア機構を用いて実装された高速な同期機構を用いるが（ステップS202, S203, S204, S205）、スレッドの状態が変化して密結合関係になくなった状態では（ステップS201のNO）、メモリ14上に実装された同期機構化あるいはVPU実行環境401のサービスとして提供されている同期機構を用いるように同期機構を切り替える（ステップS206, S207, S208）。

この切り替え手段は、VPU12上で動作するプログラムに対して、ライブラリの形式で提供するようにしても良いし、VPU12側のVPU実行環境502の提供するサービスとして提供することもできる。複数の同期機構を切り替える方式としては、あらかじめ複数の同期機構を確保しておいて、それを使い分けるようにすることもできるし、切り替えを行う時点で新しく同期機構を確保するよ

うにすることもできる。

#### 【0098】

VPU12のローカルストレージを用いた同期機構は、蜜結合スレッドグループに属するスレッド間のように、同期処理を行う時点で、同期機構を実装しているVPU12のローカルストレージが必ず有効になっている必要がある。この制限を緩和する方式としては、スレッドが実行中（RUNNING状態）で無い場合には、そのスレッドを最後に実行していたときのローカルストレージの内容をメモリに保存しておき、そのスレッドに対応付けられたローカルストレージを指すページテーブルあるいはセグメントテーブルのエントリからは、その保存したメモリ領域を指すようにマッピングを制御する。この方式により、相手のスレッドが実行中でなくても、相手のスレッドに対応付けられたローカルストレージがあたかもあるように、スレッドの実行を続けることができる。実行中で無かったスレッドが、VPU12を割り当てられて実行を始めるときには、メモリ14に保存していた内容を、再びローカルストレージに戻して、対応するページテーブルあるいはセグメントテーブルのマップを変更する。このように、VPU12のローカルストレージのバックアップコピーに対しても動作可能なように同期機構を実装しておくことで、蜜結合スレッドグループに属するスレッドでなくても、VPU12のローカルストレージを用いて実装した高速な同期機構を利用できるようになる。

#### 【0099】

（予約グラフ）

図61は、図9に例として示した処理フローに対応する予約グラフを示したものである。図61において、6つの四角い箱は実行期間（execution term）を表している。各実行期間の四角の左上の番号は予約する実行期間のIDであり、実行期間の四角の中の記号は、その実行期間に対応付けられているスレッドコンテキストの識別子である。実行期間の四角の下の数値は、その実行期間の長さ（コスト）を表している。実行期間の四角の間を結ぶ矢印は、ここではすべて先行制約を表している。すなわち、矢印が入る実行期間は、必ず矢印が出ている実行期間が終わった後で実行を開始することを示している。また、矢印に添えられてい

る番号はその矢印で結ばれた実行期間の間でデータの受け渡しに使うバッファの ID を表しており、番号と共に添えられている数値はバッファのサイズを表している。図 6 1 に示した予約グラフに従って処理を実行するための手順は、以下のようになる。

#### 【0100】

1. DEMUX プログラム 111 を実行するスレッドコンテキストを作成して、その識別子を DEMUX とする。
2. A-DEC プログラム 112 を実行するスレッドコンテキストを作成して、その識別子を A-DEC とする。
3. V-DEC プログラム 113 を実行するスレッドコンテキストを作成して、その識別子を V-DEC とする。
4. TEXT プログラム 114 を実行するスレッドコンテキストを作成して、その識別子を TEXT とする。
5. PROG プログラム 115 を実行するスレッドコンテキストを作成して、その識別子を PROG とする。
6. BLEND プログラム 116 を実行するスレッドコンテキストを作成して、その識別子を BLEND とする。

#### 【0101】

7. 図 6 2 に示すようなデータ構造の予約リクエストを作成し、VPU 実行環境 401 に渡して予約を行う。

ここで手順 1 から 6 までのスレッドコンテキストの作成は、スレッドとして実行したいプログラムを指定して VPU 実行環境 401 に依頼すると、VPU 実行環境 401 が必要な資源を割り当ててスレッドコンテキストを作成し、そのハンドルを返してくるので、それを識別子と関連付けている。

#### 【0102】

図 6 2 の予約リクエストは、BUFFER と書かれたバッファデータと、TASK と書かれた実行期間データから構成される。バッファデータは、実行期間の間でデータを受け渡すために用いるメモリ 14 上のバッファを宣言するもので、Id：にバッファ番号を、Size：にバッファサイズを、SrcTask：に

データを書き込む実行期間の番号を、`D s t T a s k :`にデータを読み出す実行期間の番号を持つ。実行期間データは、`I d :`に実行期間番号を、`C l a s s :`にスレッドクラス（`V P U`は`V P U`スレッドであることを示し、`H R T`はハードリアルタイムクラスであることを示す。他に、`M P U`スレッドを示す`M P U`や、ソフトリアルタイムクラスを示す`S R T`や、ベストエフォートクラスを示す`B S T`などがある）を、`T h r e a d C o n t e x t :`にこの実行期間に対応付けるスレッドコンテキストを、`C o s t :`にこの実行期間の長さあるいはコストを、`C o n s t r a i n t :`にこの実行期間を基準とする各種の制約を、`I n p u t B u f f e r :`にこの実行期間で読み出すバッファの識別子のリストを、`O u t p u t B u f f e r :`にこの実行期間で書き込むバッファの識別子のリストを持つ。`C o n s t r a i n t :`には、先行制約を示す`P r e c e d e n c e :`や、絶対タイミング制約を示す`A b s o l u t e T i m i n g :`や、相対タイミング制約を示す`R e l a t i v e T i m i n g :`や、排他制約を示す`E x c l u s i v e :`などを指定でき、それぞれ制約の相手になる実行期間の番号のリストを持つ。

### 【0103】

図62の予約リクエストで予約したバッファ領域は、`V P U`実行環境401が、バッファにデータを書き込むスレッドの実行開始時に割り当て、データを読み出すスレッドの実行終了時に解放する。割り当てられたバッファのアドレスは、たとえばスレッドの起動時にあらかじめ決まっているアドレスあるいは変数あるいはレジスタなどを用いて、スレッドに通知することができる。本実施形態のリアルタイム処理システムでは、図7に示したようなプログラムモジュール100が与えられたときに、その中にある、図8に示すような構成記述117を読み込んで、それに基づいて、上記の手順でスレッドコンテキストの生成と図62の予約リクエストの作成・発行を行って、そのプログラムモジュール100の実行を行う機能を提供する。この機能により、図7のようなプログラムモジュール100によって記述された専用ハードウェアの処理を、複数のプロセッサによるソフトウェア処理によって実現することが可能となる。実現したいハードウェア毎に図7のような構造を持つプログラムモジュールを作成して、それを本実施形態の

リアルタイム処理システムに準拠した機能が組み込まれた機器で実行することにより、当該機器を所望のハードウェアとして動作させることが可能となる。

#### 【0104】

図62に示す予約リクエストが与えられると、VP U実行環境401は、各実行期間を周期内のどのタイミングでどのVP U12で実行するかを決める。これがスケジューリングである。本実施形態のリアルタイム処理システムが組み込まれる電子機器の種類によっては、実際には、このような予約リクエストが同時に複数与えられることもあるので、それらが矛盾ないように（与えられた制約が満たされないことがないように）処理のタイミングが決定される。例えば、図63に示すように、VP U12が2つあるときに、図62の予約リクエストだけが入っていたとすると、DEMUX、V-DEC、PROG、BLENDの並行に実行できない処理をVP U0で順次実行し、DEMUXの実行後に並行して動作できるA-DECとTEXTをVP U1で実行するようにスケジューリングする。

#### 【0105】

（ソフトウェアパイプライン）

ここでもし、ひとつの周期内でDEMUX、V-DEC、PROG、BLENDを順次実行できるだけの時間がない場合には、複数の周期にまたがるようにソフトウェアパイプライン化を行う。例えば図64に示すように、最初の周期1ではDEMUXとV-DECをVP U0で行い、次の周期2でA-DEC、TEXT、PROG、BLENDの処理をVP U1で行うようにする。この周期2では、A-DEC、TEXT、PROG、BLENDの処理と並行して、次のフレームのDEMUXとV-DECがVP U0によって実行される。すなわち、図65に示すように、VP U0がDEMUXとV-DECを実行している間、VP U1では前の周期のDEMUXとV-DECの出力を受けたA-DEC、TEXT、PROG、BLENDが動くという具合に、パイプライン処理を行う。

#### 【0106】

（バッファ量を考慮したスケジューリング）

ある実行期間に実行するスレッドと、別の実行期間に実行するスレッドの間で、バッファを使ってデータを送る場合、そのバッファはデータを書き込む側の実



行期間の初めから、データを読み出す側の実行期間の最後までの間、専有されることになる。例えば、図 6 6 に示すように、実行期間 A と実行期間 B の間でバッファを使ってデータを送る場合、図 6 6 に示すように、実行期間 A の初めから、実行期間 B の最後までの間、メモリ 14（主記憶）上のバッファは占有して使用されることになる。そのため、実行期間 A から実行期間 B にバッファを使ってデータを送るときで、ソフトウェアパイプライン化した際に実行期間 A と B が隣り合う別の周期で実行するような場合、実行期間 A と B の実行タイミングによって必要なバッファの量が変わってくる。例えば、図 6 7 に示すように、各周期内で実行期間 A が B よりも早く実行されるようにスケジューリングした場合、実行期間  $A_n$ （ $A_n$  は周期  $n$  における A をあらわす）からのデータは次の周期の実行期間  $B_n$  に渡され、実行期間  $A_{n+1}$  からのデータは次の周期の実行期間  $B_{n+1}$  に渡される。このとき、実行期間  $A_{n+1}$  は  $A_n$  と  $B_n$  に挟まれているため、 $A_n$  が  $B_n$  にデータを渡すために使っているバッファは利用できず、新しいバッファを用いる。つまり、ダブルバッファリングが必要になる。一方、図 6 8 に示すように、周期内で実行期間 A が B の終了後に開始するようにすると、実行期間  $A_n$  がデータを書いたバッファを  $B_n$  が読んだ後、同じバッファを使いまわして、 $A_{n+1}$  がデータを書いて  $B_{n+1}$  が読むように、シングルバッファですむ。

#### 【0107】

本実施形態のリアルタイム処理システムでは、VPU 実行環境 401 のスケジューラが、このように、バッファメモリ領域の使用量ができるだけ少なくなるように、予約される実行期間をスケジューリングする。すなわち、ソフトウェアパイプラインを行う場合には、図 6 9 のフローチャートに示されているように、VPU 実行環境 401 のスケジューラは、VPU 0, 1 の 2 つの VPU によるソフトウェアパイプラインを実行するために、一連の処理を 2 つの部分処理（VPU 0 によって先行して実行される部分処理と、その部分処理に後続して VPU 1 によって実行される部分処理）に分割する（ステップ S211）。そして、VPU 実行環境 401 のスケジューラは、2 つの部分処理間でバッファを介して入出力を行うスレッド同士（例えば、先行して実行される部分処理内のスレッド A と、後続して実行される部分処理内のスレッド B）を抽出し（ステップ S212）、

各周期において、先行して実行される部分処理内のスレッドAが、後続する部分処理内のスレッドBの実行期間終了後に開始されるように、スレッドA、Bをスケジューリングする（ステップS213）。

#### 【0108】

（階層構造を持つ予約グラフ）

図61に示した予約グラフは階層構造を持っていないが、図70に示すように、階層構造を持つ予約グラフを扱うことも出来る。図70の例では、実行期間AはBに先行し、BはCに先行する。Bの中はDがEとFに先行している。それゆえ、階層を解くと、AはDに先行し、EとFはCに先行することになる。

#### 【0109】

（密結合スレッドグループを考慮した予約リクエスト）

例えば図61に示した予約グラフにおいてV-DECを実行するスレッドとPROGを実行するスレッドが密結合スレッドグループに属する場合、その結合属性を示す予約リクエストが図71のように生成される。図71の例においては、TightlyCoupled：に相手先のスレッドに対応する実行期間のIDが記述されている。これにより、V-DECを実行するスレッドとPROGを実行するスレッドがそれぞれ別のVPUによって同時に実行されるように、例えば図72に示すようにスケジューリングされる。この場合、V-DECを実行するスレッドとPROGを実行するスレッドとの間の通信はローカルストレージを介して実行できるので、バッファをメモリ14上に用意する必要はない。

#### 【0110】

（構成記述に基づくスケジューリングアルゴリズム）

以下、プログラムモジュールに組み込まれた構成記述に基づいて各スレッドの実行期間を予約するための処理手順について説明する。

#### 【0111】

図7のプログラムモジュール100内の構成記述117は、図8の例のようになっている。この構成記述117が与えられると、VPU実行環境401は次の手順を実行する。

1. 構成記述117のモジュール欄に書かれている各プログラムをロードして

、それぞれを実行するスレッドを生成する。このとき、本実施形態では、構成記述 117 のエントリそれぞれに対して一つのスレッドを生成する。構成記述 117 の中に、同じモジュール名を持つ複数のエントリが存在する場合には、同じモジュールを実行する複数のスレッドをそれぞれのエントリと対応するように生成することになる。なお、図 8 の例では、すべてのスレッドはひとつのプロセスに属するように生成されるものとしているが、それぞれのスレッドが別のプロセスに属するように実施することもできるし、あるグループのスレッドはあるプロセスに属し、また他のグループのスレッドは別のプロセスに属するといったように実施することもできる。

2. 構成記述 117 の情報から、図 62 で説明したような予約リクエストのデータ構造を作成する。

3. 予約リクエストを VPU 実行環境に渡して処理のスケジューリングを行い、実行を開始する。

#### 【0112】

この 2 番目の予約リクエストを作成するステップは、次のように行う。

まず、構成記述 117 の出力欄に 1 対 1 に対応するように、BUFFER レコードを作成して予約リクエストに加える。例えば、図 8 の構成記述 117 の例では、DEMUX モジュールの 2 番目の出力は 1MB のバッファを使ってデータを VDEC に渡しているので、それに対応するように、図 62 の Id が 2 の BUFFER レコードを作成している。Id が 2 の BUFFER レコードには、そのバッファサイズが Size 欄に 1MB と記録され、そのバッファにデータを書き込む DEMUX モジュールに対応するタスクである Id が 1 の TASK レコードへの参照が Src Task 欄に記録され、そのバッファのデータを読み出す VDEC モジュールに対応するタスクである Id が 3 の TASK レコードへの参照が Dst Task 欄に記録されている。

#### 【0113】

次に、構成記述 117 のモジュール欄に 1 対 1 に対応するように、TASK レコードを作成して予約リクエストに加える。例えば、図 8 の構成記述 117 の例で、VDEC モジュールに対応する TASK レコードとして、図 62 の Id が

3のTASKレコードを作成している。Idが3のTASKレコードには、以下のような情報が記録されている。

#### 【0114】

**Class欄:** このTASKレコードに指定されるスレッドをどのような属性で実行させるかを示すフラグ。VPUはVPU上で実行するスレッドであることを、HRTはハードリアルタイムクラスのスレッドであることを示す。これらの情報は、図8の例では構成記述117のスレッドパラメタに記述されている情報をもとに設定する。

#### 【0115】

**ThreadContext欄:** このTASKレコードで実行の予約を行いたいスレッドのスレッドコンテキストを指定する。具体的には、図8の構成記述117のモジュール欄に指定されたプログラムモジュールをロードして、それを実行するスレッドをVPU実行環境401によって生成し、そのスレッドのスレッドコンテキストの識別子（あるいはポインタなど）を、ThreadContext欄に記録する。

**Constraint欄:** このTASKレコードに関する制約条件を記録する。先行制約の場合は、Precede:の後にそのTASKが先行する他のTASKのIdを必要な数指定する。Idが3のTASKレコードの場合、Idが5のPROGモジュールに対応するTASKに先行することを示している。

**InputBuffer欄:** このTASKレコードで指定されるスレッドがデータを読み出すバッファのBufferレコードのIdを必要な数指定する。

**OutputBuffer欄:** このTASKレコードで指定されるスレッドがデータを書き込むバッファのBufferレコードのIdを必要な数指定する。

このようにして、構造記述が与えられるとそれに対する予約リクエストが作成される。

#### 【0116】

次に、その予約リクエストをVPU実行環境401内のスケジューラに渡すと

、スケジューラは、指定された予約リクエストを実行するのに必要なスケジュールを作成する。このスケジューリング処理の結果作成されたスケジュールは、例えば図 6 3 に示すような、各周期のどのタイミングで、どの V P U を、どれだけの時間、どのスレッドに割り当てるかを示すものである。実際には、例えば図 7 3 のような予約リストによって表現されるように実施することができる。

#### 【 0 1 1 7 】

図 7 3 の予約リストは、各 V P U に対応付けられた予約エントリから構成される。予約エントリには、ひとつのスレッドに対して、それを各周期内のどのタイミングで V P U を割り当てて実行を始めるかを開始時間欄に、どれくらいの時間で V P U を取り上げるかを実行期間欄に、そのスレッドの識別子を実行スレッド欄に記録している。それらの予約エントリは、実行する V P U 別に、開始時間順にソートされて予約リストにつながれている。

#### 【 0 1 1 8 】

図 6 2 または図 7 1 に示すような予約リクエストから、図 7 3 に示すような予約リストを作成する手順は、例えば図 7 4 のフローチャートで示す手順で実施できる。

基本は、予約リクエスト中の各 T A S K レコードを、B U F F E R を使った入出力関係を考慮して順序付けして、データの流れる順に、V P U の実行時間を割り付けて行けばよい。このとき、密結合スレッドグループに指定されている T A S K 群には、それぞれの T A S K のスレッドに同時に V P U を割り付けるようにする必要がある。

#### 【 0 1 1 9 】

図 7 4 にその手順を示す。予約リクエストが与えられると、その中の T A S K レコードに指定されているすべてのタスクの集合に対して、以下の手順でスケジューリング（いいかえると、スケジュールの割り付け、あるいは予約リストの作成）を行う。

ステップ S 3 0 1： 全ての入力タスクが割り付け済みのタスクで、密結合指定のないタスクを選択する。すなわち、未割り付けのタスク（すなわち、まだ予約エントリを作って予約リストにつないでいないタスク）の中で、そのタスクの入

力となるデータのソースになるタスクがすべて割り付け済み（予約エントリが予約リストに入っている）であるか、あるいはそのタスクはデータの入力を持たない場合であって、かつ、そのタスクが密結合指定されていないものが存在すれば、それを選択してステップ S 3 0 2 へ、そうでなければステップ S 3 0 4 へ行く。

ステップ S 3 0 2： 選択したタスクを予約できる V P U が存在すれば（言い換えると、他のタスクとの間の制約を満たす開始時間と実行期間を予約できる V P U が存在すれば）、ステップ S 3 0 3 へ、そうでなければスケジューリング不可能なので失敗を通知する。

ステップ S 3 0 3： 選択したタスクの予約エントリを作成して、予約リストにつなぐ。

#### 【 0 1 2 0 】

ステップ S 4 0 4： 全ての入力タスクが割り付け済みのタスクで、密結合関係にあるタスク群を選択する。すなわち、未割り付けのタスク（すなわち、まだ予約エントリを作って予約リストにつないでいないタスク）の中で、そのタスクの入力となるデータのソースになるタスクがすべて割り付け済み（予約エントリが予約リストに入っている）であるか、あるいはそのタスクはデータの入力を持たないものの集合であって、かつ、その集合に属するタスク間が密結合指定されているものが存在すれば、そのタスク集合（タスク群とも呼ぶ）を選択してステップ S 3 0 5 へ、そうでなければ既にすべてのタスクを割り付けているのでスケジューリング処理を終了する。

ステップ S 3 0 5： 選択したタスク集合に含まれるすべてのタスクを同時に（同じ開始時間で同じ実行期間を持つように）予約できる複数の V P U が存在すればステップ S 3 0 6 へ、そうでなければスケジューリング不可能なので失敗を通知する。

ステップ S 3 0 6： 選択したタスク集合のすべてのタスクの予約エントリを作成して、予約リストにつなぐ。

#### 【 0 1 2 1 】

ここでの説明はひとつの予約リクエストのスケジューリングの手順について述

べたが、上述したように、実際は、ひとつのシステムにおいて複数の予約リクエストが同時に存在することが普通である。そのような場合には、複数の予約リクエストを順次上記の手順でスケジューリングするように実施することもできるし、より望ましくは、同時に複数の予約リクエストを上記の手順でスケジューリングするように実施する。

#### 【0 1 2 2】

以上、デジタルテレビ放送用受信機の動作を記述したプログラムモジュールを例に説明したが、他の様々なハードウェアの動作を記述したプログラムモジュールを用意することにより、デジタルテレビ放送用受信機以外の他の任意のハードウェアの動作をソフトウェアによって実現することが出来る。

#### 【0 1 2 3】

なお、図 1 の計算機システムに設けられたMPU 1 1 と複数のVPU 1 2 は。それらを 1 チップ上に混載した並列プロセッサとして実現することもできる。この場合も、MPU 1 1 によって実行されるVPU 実行環境、あるいは特定の一つのVPUなどによって実行されるVPU 実行環境が、複数のVPU 1 2 に対するスケジューリングを制御することが出来る。

#### 【0 1 2 4】

またVPU 実行環境として動作するプログラムまたはそのVPU 実行環境を含むオペレーティングシステムなどのプログラムをコンピュータ読み取り可能な記憶媒体に記憶することにより、その記憶媒体を通じて当該プログラムを、ローカルプロセッサをそれぞれ有する複数のプロセッサを含むコンピュータに導入して実行するだけで、本実施形態と同様の効果を得ることが出来る。

#### 【0 1 2 5】

また、本発明は上記実施形態そのままに限定されるものではなく、実施段階ではその要旨を逸脱しない範囲で構成要素を変形して具体化できる。また、上記実施形態に開示されている複数の構成要素の適宜な組み合わせにより、種々の発明を形成できる。例えば、実施形態に示される全構成要素から幾つかの構成要素を削除してもよい。さらに、異なる実施形態にわたる構成要素を適宜組み合わせてもよい。

## 【0126】

## 【発明の効果】

以上説明したように、本発明によれば、互いに協調して動作するスレッド間のデータの受け渡しのような相互作用を効率よく実行することが可能となる。

## 【図面の簡単な説明】

【図1】 本発明の一実施形態に係るリアルタイム処理システムを構成する計算機システムの例を示すブロック図。

【図2】 同実施形態のリアルタイム処理システムに設けられたMPUおよびVPUそれぞれの構成を示すブロック図。

【図3】 同実施形態のリアルタイム処理システムで用いられる仮想アドレス変換機構の例を示す図。

【図4】 同実施形態のリアルタイム処理システムにおける実アドレス空間にマッピングされるデータの例を示す図。

【図5】 同実施形態のリアルタイム処理システムにおける実効アドレス空間、仮想アドレス空間、実アドレス空間を説明するための図。

【図6】 デジタルテレビ放送の受信機の構成を示すブロック図。

【図7】 同実施形態のリアルタイム処理システムによって実行されるプログラムモジュールの構成の例を示す図。

【図8】 図7のプログラムモジュール内に含まれる構成記述の例を示す図。

【図9】 図7のプログラムモジュールに対応するプログラム間のデータの流れを示す図。

【図10】 図7のプログラムモジュールが2つのVPUによって並列に実行される様子を示す図。

【図11】 図7のプログラムモジュールが2つのVPUによってパイプライン形式で実行される様子を示す図。

【図12】 同実施形態のリアルタイム処理システムにおけるオペレーティングシステムの実装形態の例を示す図。

【図13】 同実施形態のリアルタイム処理システムにおけるオペレーティ



ングシステムの実装形態の他の例を示す図。

【図 14】 同実施形態のリアルタイム処理システムにおける仮想計算機 OS とゲスト OS との関係を示す図。

【図 15】 同実施形態のリアルタイム処理システムにおいて複数のゲスト OS に時分割で資源が割り当てられる様子を示す図。

【図 16】 同実施形態のリアルタイム処理システムにおいてある特定のゲスト OS によって特定の資源が専有される様子を示す図。

【図 17】 同実施形態のリアルタイム処理システムにおいてスケジューラとして用いられる VPU 実行環境を示す図。

【図 18】 同実施形態のリアルタイム処理システムで用いられる仮想計算機 OS に VPU 実行環境を実装した例を示す図。

【図 19】 同実施形態のリアルタイム処理システムで用いられる一つのゲスト OS として VPU 実行環境を実装する例を示す図。

【図 20】 同実施形態のリアルタイム処理システムで用いられる複数のゲスト OS それぞれに VPU 実行環境を実装する例を示す図。

【図 21】 同実施形態のリアルタイム処理システムで用いられる一つのゲスト OS に VPU 実行環境を実装する例を示す図。

【図 22】 同実施形態のリアルタイム処理システムで用いられる MPU 側 VPU 実行環境と VPU 側 VPU 実行環境を説明するための図。

【図 23】 同実施形態のリアルタイム処理システムで用いられる VPU 側 VPU 実行環境によって実行される処理手順を示すフローチャート。

【図 24】 同実施形態のリアルタイム処理システムで用いられる MPU 側 VPU 実行環境によって実行される処理手順を示すフローチャート。

【図 25】 同実施形態のリアルタイム処理システムにおいて密結合スレッドグループに属するスレッド群がそれぞれ別のプロセッサによって同時に実行される様子を示す図。

【図 26】 同実施形態のリアルタイム処理システムにおける密結合スレッド間の相互作用を説明するための図。

【図 27】 同実施形態のリアルタイム処理システムにおいて各密結合スレ

ッドの実効アドレス空間に相手のスレッドが実行される V P U のローカルストレージがマッピングされる様子を示す図。

【図 2 8】 同実施形態のリアルタイム処理システムにおける疎結合スレッドグループに属するスレッド群に対するプロセッサの割り当てを説明するための図。

【図 2 9】 同実施形態のリアルタイム処理システムにおける疎結合スレッド間の相互作用を説明するための図。

【図 3 0】 同実施形態のリアルタイム処理システムにおけるプロセスとスレッドとの関係を説明するための図。

【図 3 1】 同実施形態のリアルタイム処理システムにおけるスケジューリング処理の手順を示すフローチャート。

【図 3 2】 同実施形態のリアルタイム処理システムにおけるローカルストレージのマッピングに関する第 1 の問題を説明するための図。

【図 3 3】 同実施形態のリアルタイム処理システムにおける物理 V P U と論理 V P U との関係を示す図。

【図 3 4】 同実施形態のリアルタイム処理システムにおけるローカルストレージのマッピングに関する第 2 の問題を説明するための図。

【図 3 5】 同実施形態のリアルタイム処理システムにおける実効アドレス空間共有モデルを示す図。

【図 3 6】 同実施形態のリアルタイム処理システムにおける仮想アドレス空間共有モデルを示す図。

【図 3 7】 同実施形態のリアルタイム処理システムにおける非共有モデルを示す図。

【図 3 8】 同実施形態のリアルタイム処理システムにおけるローカルストレージのマッピング変更を説明するための第 1 の図。

【図 3 9】 同実施形態のリアルタイム処理システムにおけるローカルストレージのマッピング変更を説明するための第 2 の図。

【図 4 0】 同実施形態のリアルタイム処理システムにおけるローカルストレージのマッピング変更を説明するための第 3 の図。

【図 4 1】 同実施形態のリアルタイム処理システムにおけるローカルストレージのマッピング変更を説明するための第 4 の図。

【図 4 2】 同実施形態のリアルタイム処理システムにおけるローカルストレージのマッピング変更を説明するための第 5 の図。

【図 4 3】 同実施形態のリアルタイム処理システムにおいてローカルストレージのマッピング変更を行うために実行されるアドレス管理処理の手順を示すフローチャート。

【図 4 4】 同実施形態のリアルタイム処理システムにおいて実行されるローカルストレージとメモリとの間のマッピング変更を説明するための図。

【図 4 5】 同実施形態のリアルタイム処理システムにおいて実行されるローカルストレージとメモリとの間のマッピング変更処理の手順を示すフローチャート。

【図 4 6】 同実施形態のリアルタイム処理システムにおけるスレッドの状態遷移を示す図。

【図 4 7】 同実施形態のリアルタイム処理システムにおけるスレッドと実効期間との関係を説明するための図。

【図 4 8】 同実施形態のリアルタイム処理システムにおける密結合スレッド群がある実効期間において同時に実行される様子を示す図。

【図 4 9】 同実施形態のリアルタイム処理システムにおける周期実行モデルを示す図。

【図 5 0】 同実施形態のリアルタイム処理システムにおける非周期実行モデルを示す図。

【図 5 1】 タスクグラフを説明するための図。

【図 5 2】 同実施形態のリアルタイム処理システムで用いられる予約グラフの原理を説明するための図。

【図 5 3】 同実施形態のリアルタイム処理システムで用いられる予約グラフの例を説明するための図。

【図 5 4】 同実施形態のリアルタイム処理システムで用いられる階層型スケジューラを説明するための図。

【図 5 5】 同実施形態のリアルタイム処理システムがハードリアルタイムクラスのスケジューリングのために使用するパラメータの例を説明する図。

【図 5 6】 同実施形態のリアルタイム処理システムで用いられる絶対タイミング制約を説明する図。

【図 5 7】 同実施形態のリアルタイム処理システムで用いられる相対タイミング制約を説明する図。

【図 5 8】 同実施形態のリアルタイム処理システムで用いられる相互排他制約を説明する図。

【図 5 9】 同実施形態のリアルタイム処理システムにおける同期機構を説明するための図。

【図 6 0】 同実施形態のリアルタイム処理システムにおいて同期機構を使い分ける手順を示すフローチャート。

【図 6 1】 同実施形態のリアルタイム処理システムにおいて用いられる予約グラフの例を示す図。

【図 6 2】 同実施形態のリアルタイム処理システムにおいて生成される予約リクエストの例を示す図。

【図 6 3】 同実施形態のリアルタイム処理システムが図 6 2 の予約リクエストに基づいて実行するスケジューリングの例を示す図。

【図 6 4】 同実施形態のリアルタイム処理システムによって実行されるソフトウェアパイプライン形式のスケジューリングを説明するための第 1 の図。

【図 6 5】 同実施形態のリアルタイム処理システムによって実行されるソフトウェアパイプライン形式のスケジューリングを説明するための第 2 の図。

【図 6 6】 同実施形態のリアルタイム処理システムにおいて実行されるバッファ量を考慮したスケジューリングを説明するための第 1 の図。

【図 6 7】 同実施形態のリアルタイム処理システムにおいて実行されるバッファ量を考慮したスケジューリングを説明するための第 2 の図。

【図 6 8】 同実施形態のリアルタイム処理システムにおいて実行されるバッファ量を考慮したスケジューリングを説明するための第 3 の図。

【図 6 9】 同実施形態のリアルタイム処理システムにおいて実行されるバ

ッファ量を考慮したスケジューリング処理の手順を示すフローチャート。

【図 7 0】 同実施形態のリアルタイム処理システムにおいて用いられる階層構造を持つ予約グラフの例を示す図。

【図 7 1】 同実施形態のリアルタイム処理システムによって生成される、密結合スレッドグループを考慮した予約リクエストの例を示す図。

【図 7 2】 同実施形態のリアルタイム処理システムが図 7 1 の予約リクエストに基づいて行うスケジューリングの例を示す図。

【図 7 3】 同実施形態のリアルタイム処理システムにおいて用いられる予約リストの例を示す図。

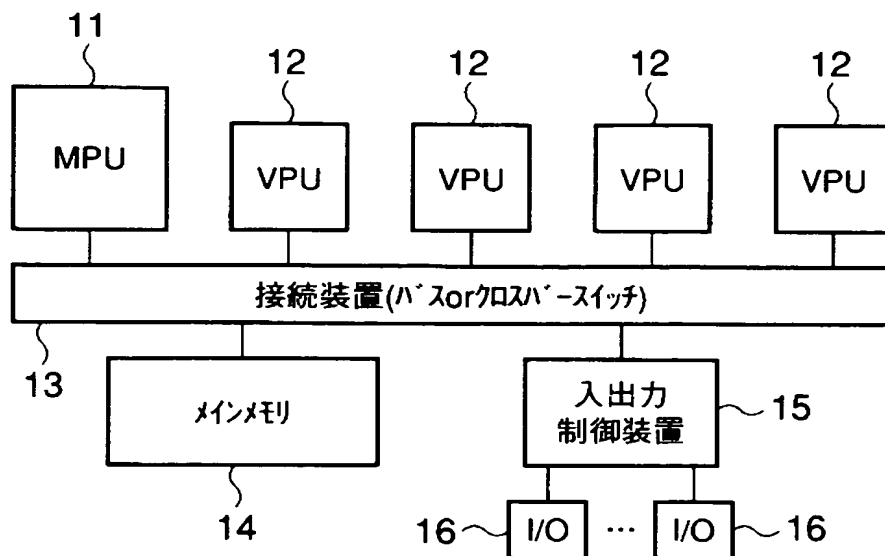
【図 7 4】 同実施形態のリアルタイム処理システムにおける実行期間予約処理の手順を示すフローチャート。

【符号の説明】

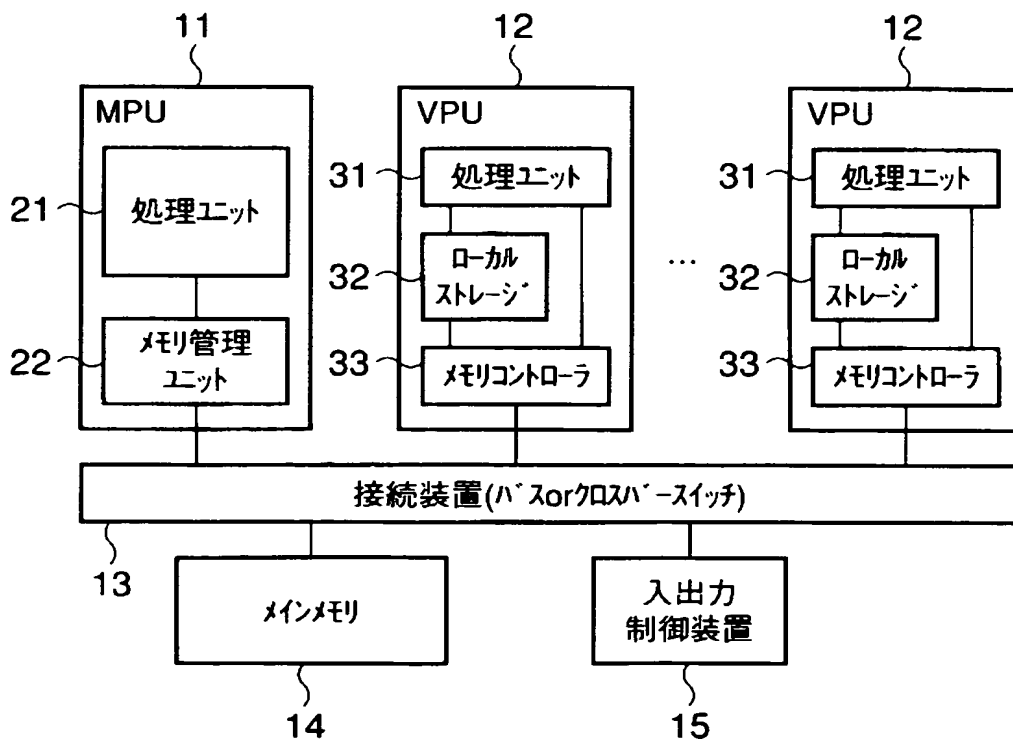
1 1…M P U (Master Processing Unit) 、 1 2…V P U (Slave Processing Unit) 、 1 4…メインメモリ、 2 1…処理ユニット、 2 2…メモリ管理ユニット、 3 1…処理ユニット、 3 2…ローカルストレージ、 3 3…メモリコントローラ、 5 0…セグメントテーブル、 6 0…ページテーブル、 1 0 0…プログラムモジュール、 1 1 7…構成記述、 3 3 1…アドレス変換ユニット、 4 0 1…V P U 実行環境。

【書類名】 図面

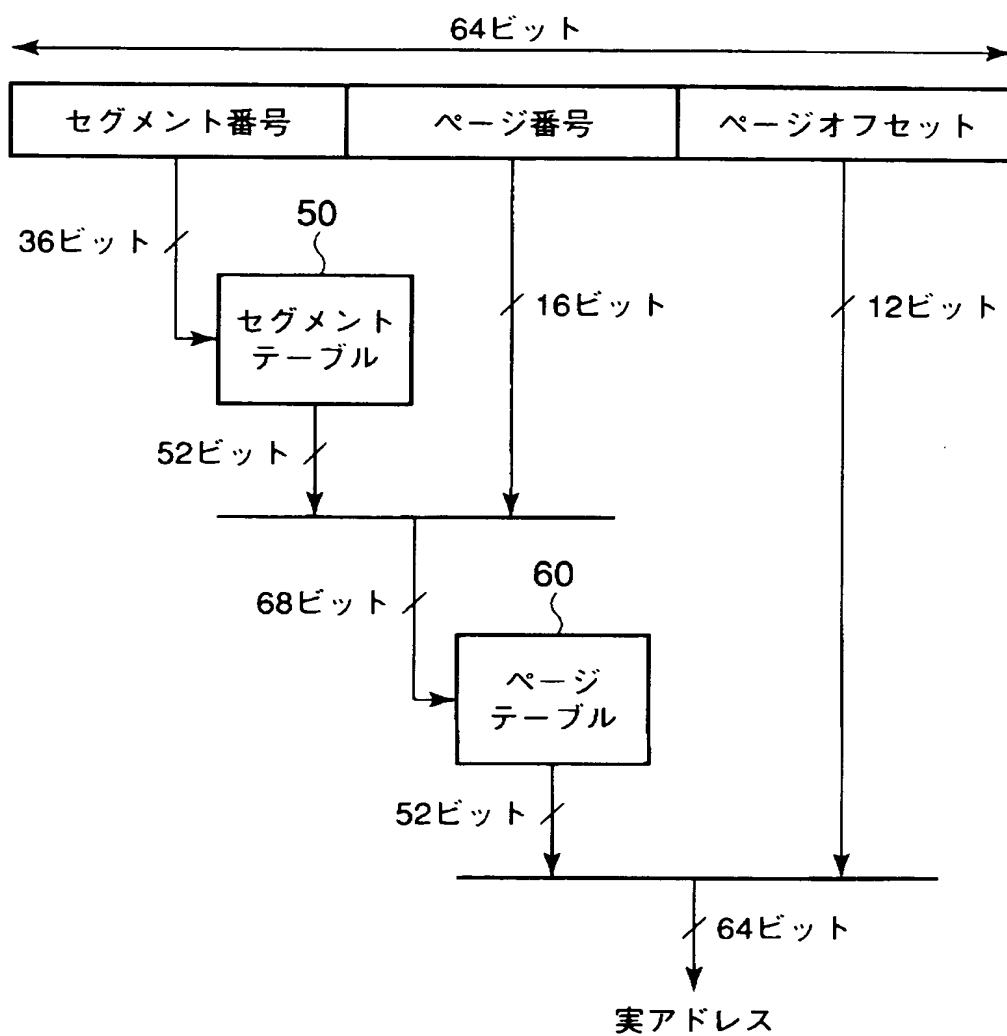
【図 1】



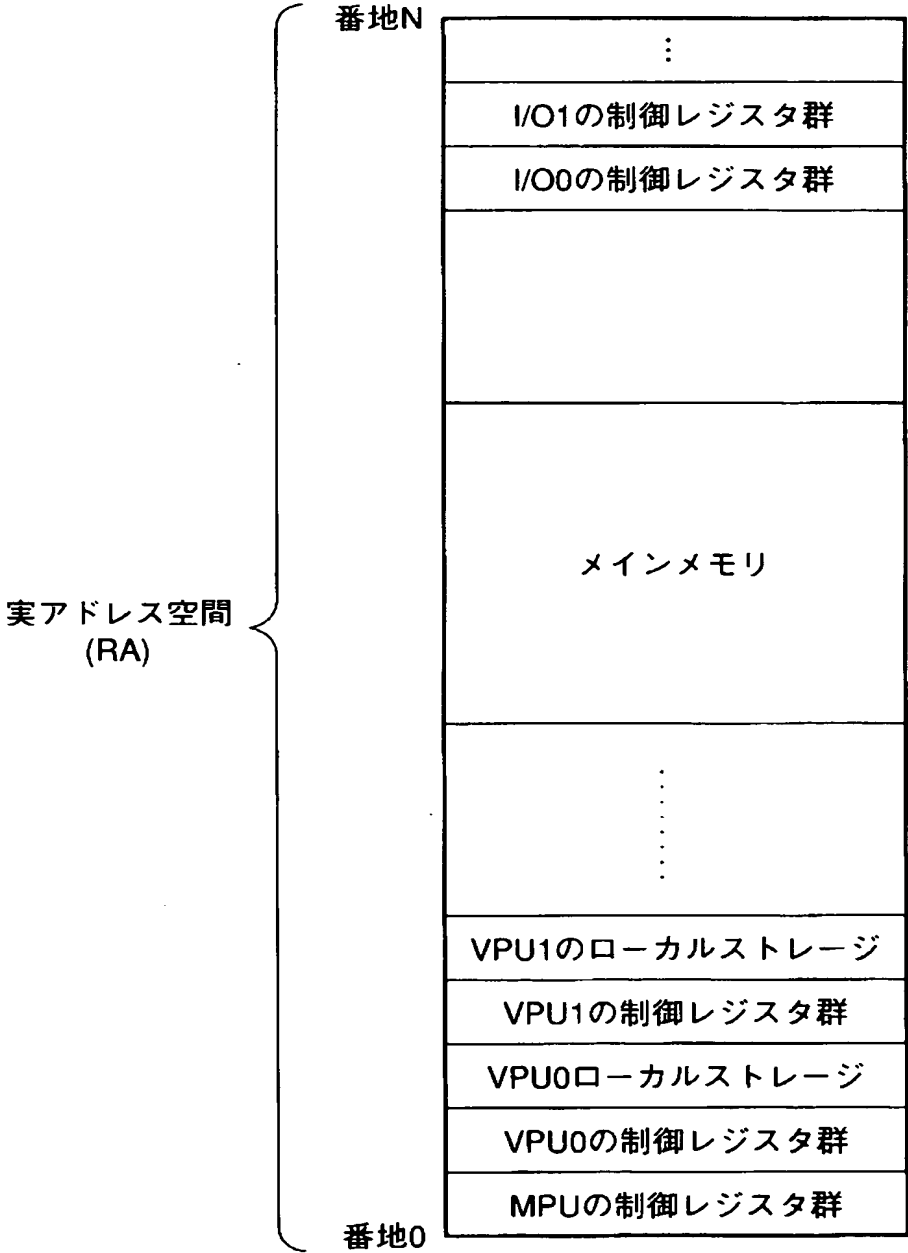
【図 2】



【図 3】

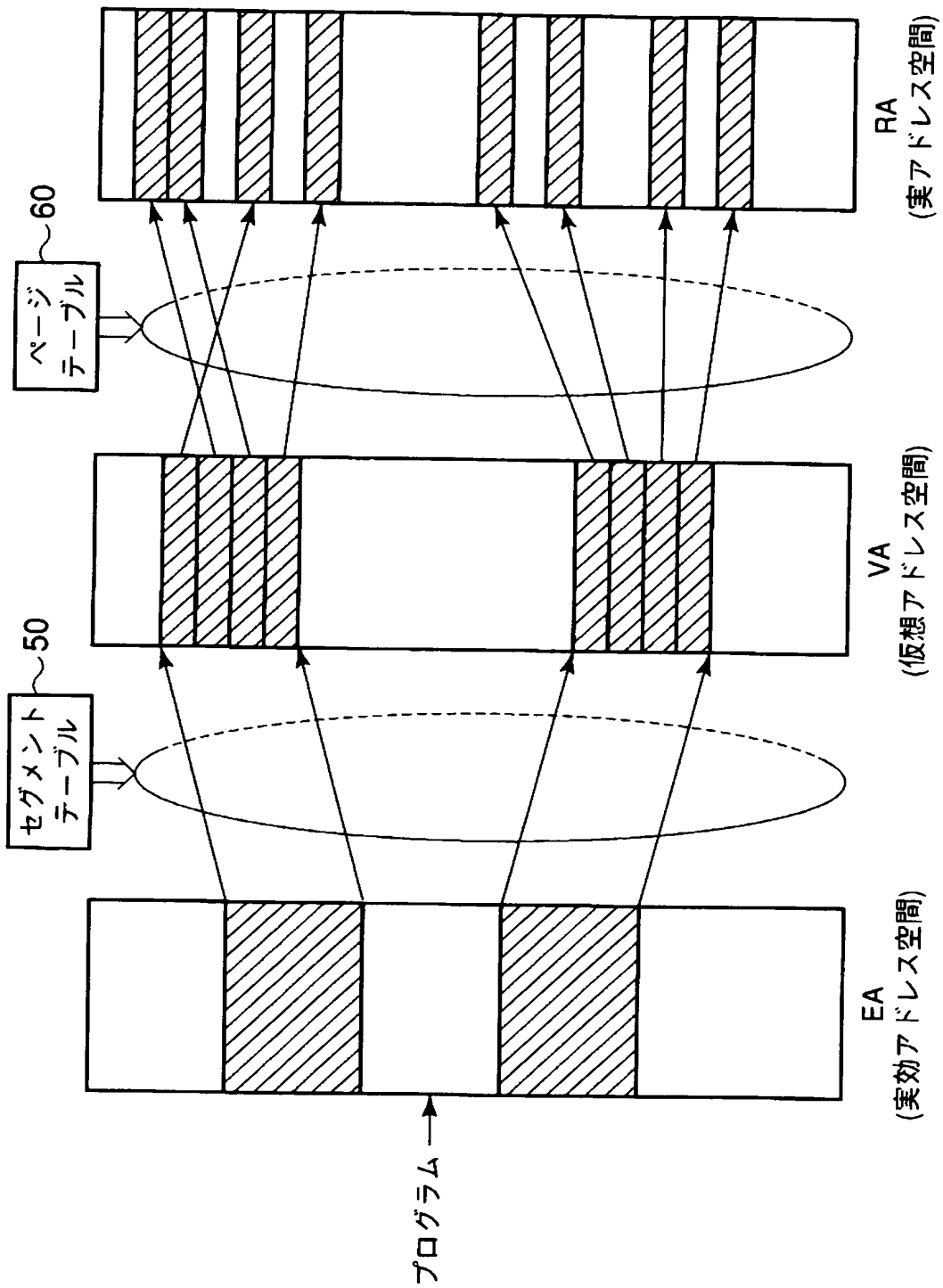


【図 4】

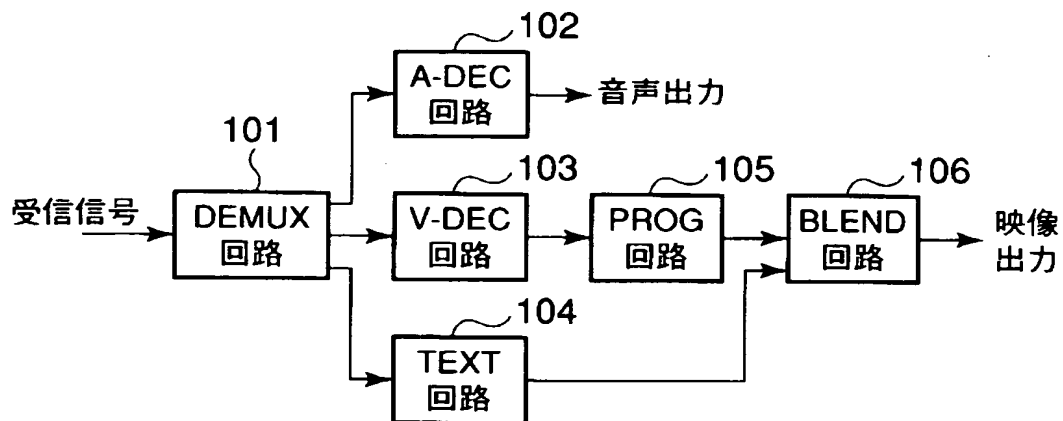




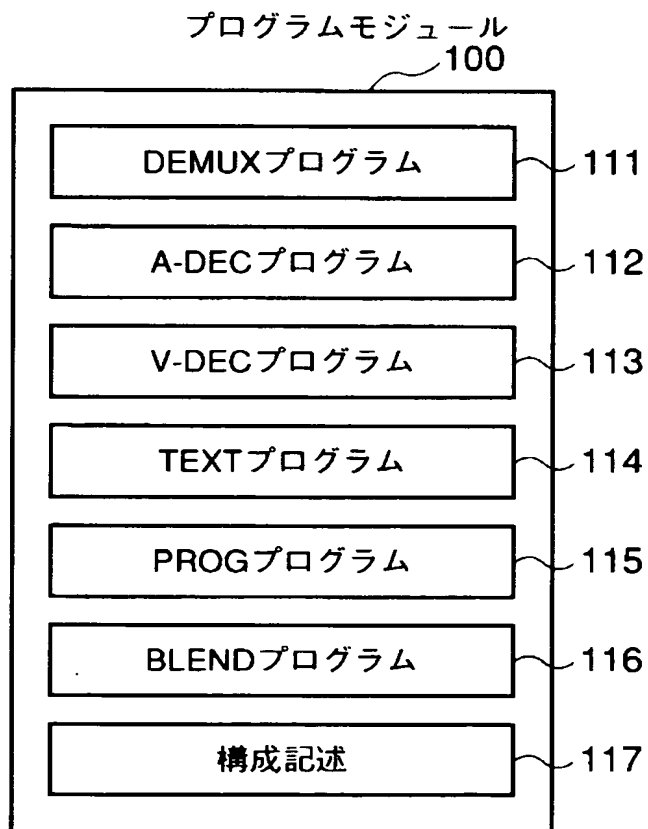
【図 5】



【図 6】



【図 7】

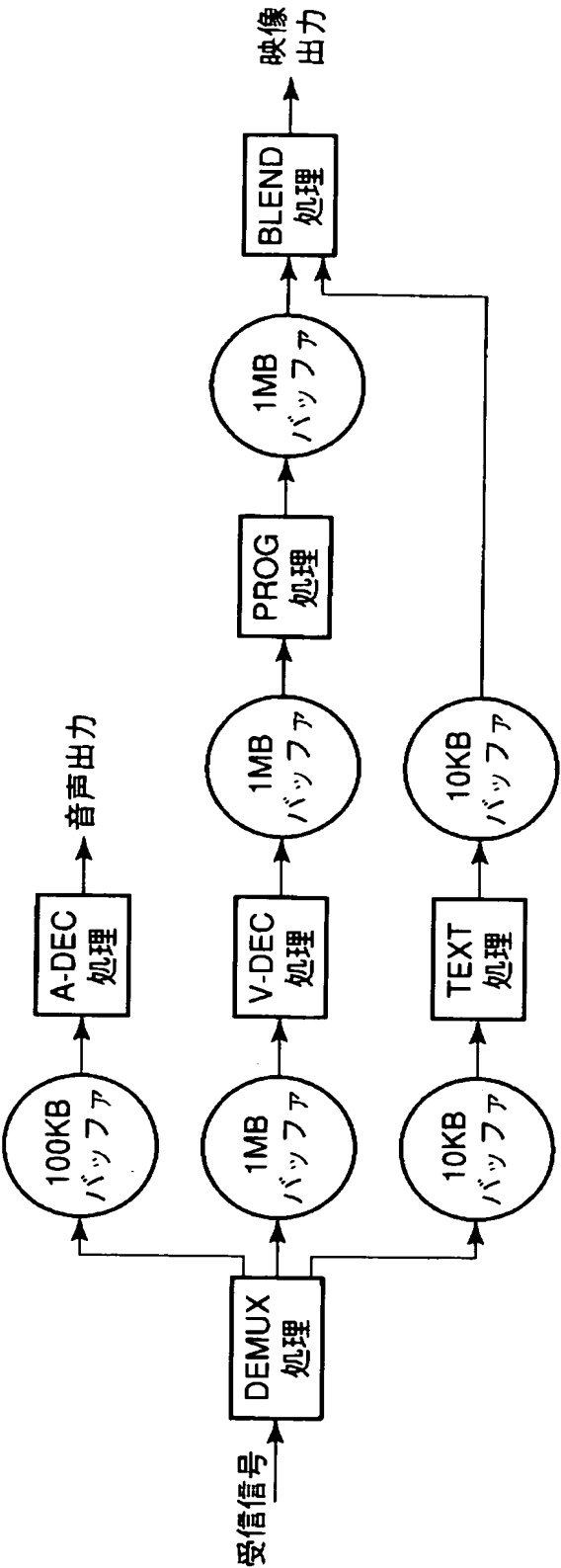


【図 8】

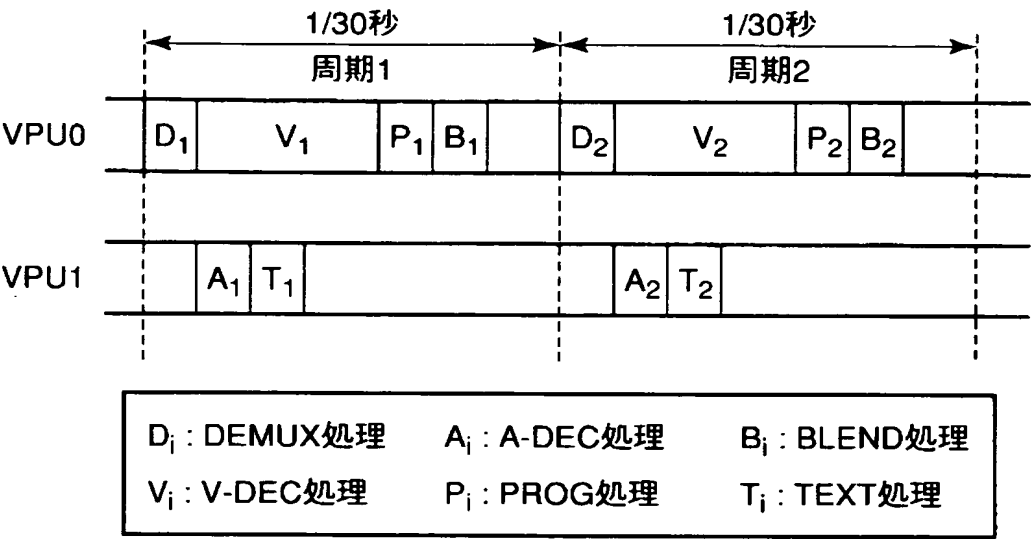
117 構成記述の例

番号	プログラム	入力	出力	コスト	バッファ
①	DEMUX	受信信号	② ③ ④	5	100KB 1MB 10KB
②	A-DEC	①	音声出力	10	——
③	V-DEC	①	⑤	50	1MB
④	TEXT	①	⑥	5	10KB
⑤	PROG	③	⑥	20	1MB
⑥	BLEND	④ ⑤	映像出力	10	——
スレッドパラメータ					
その他					

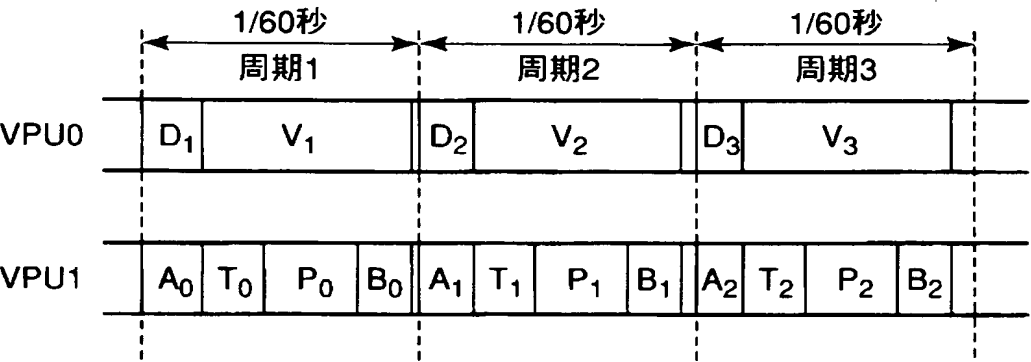
【図 9】



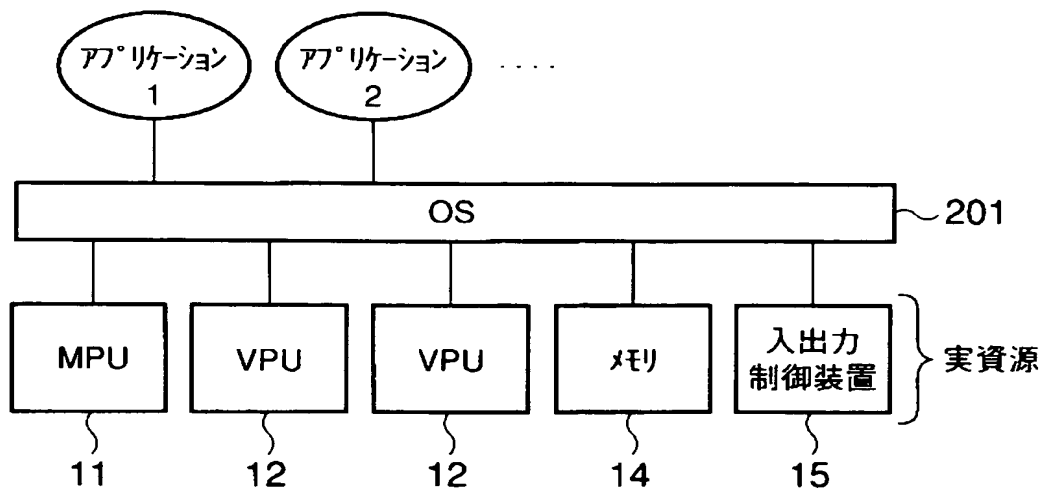
【図 1 0】



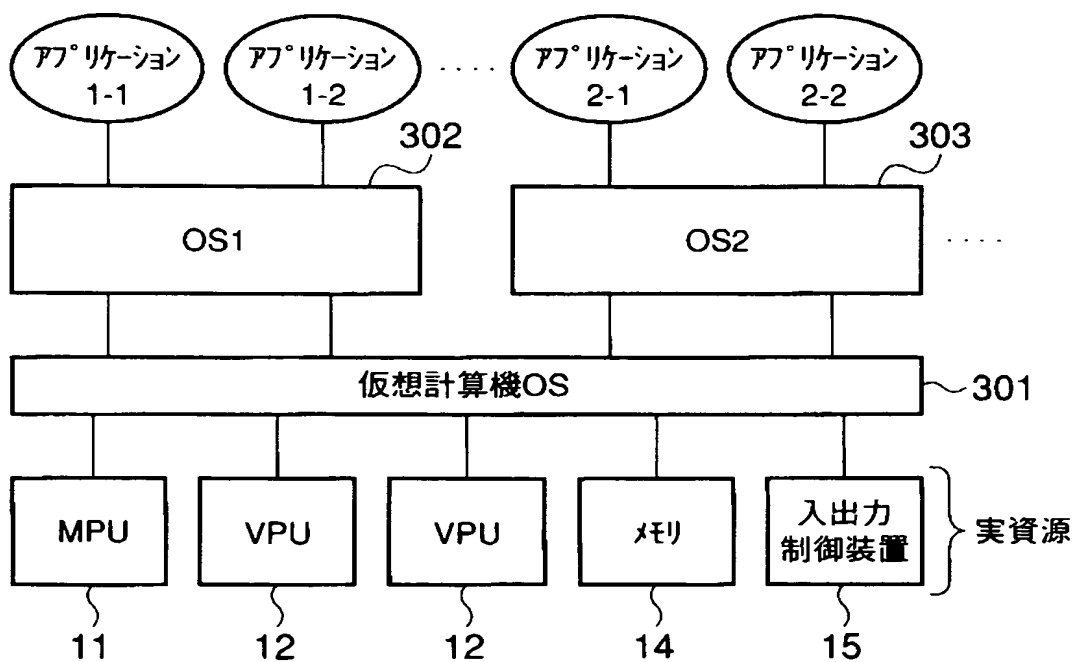
【図 1 1】



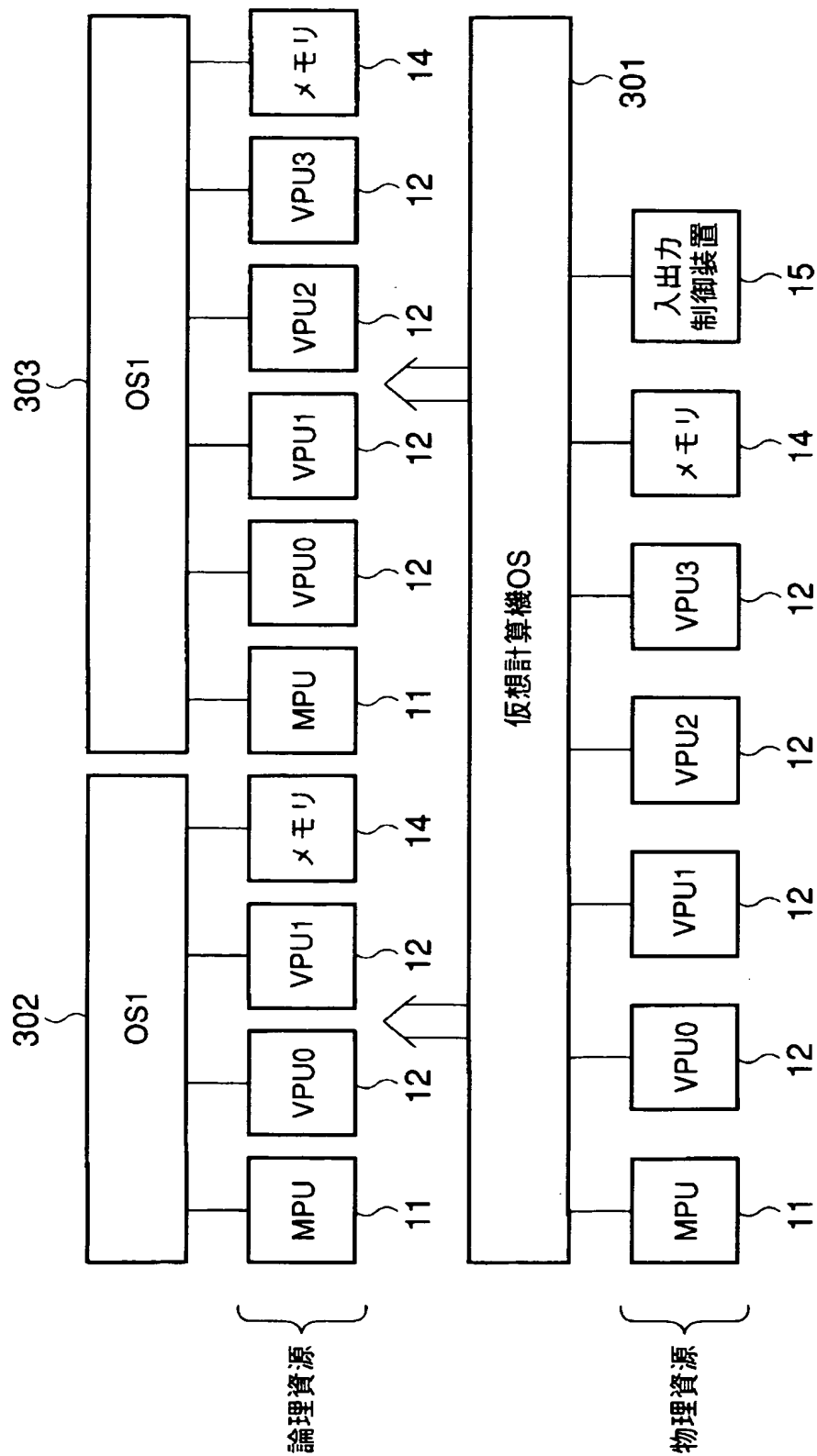
【図 12】



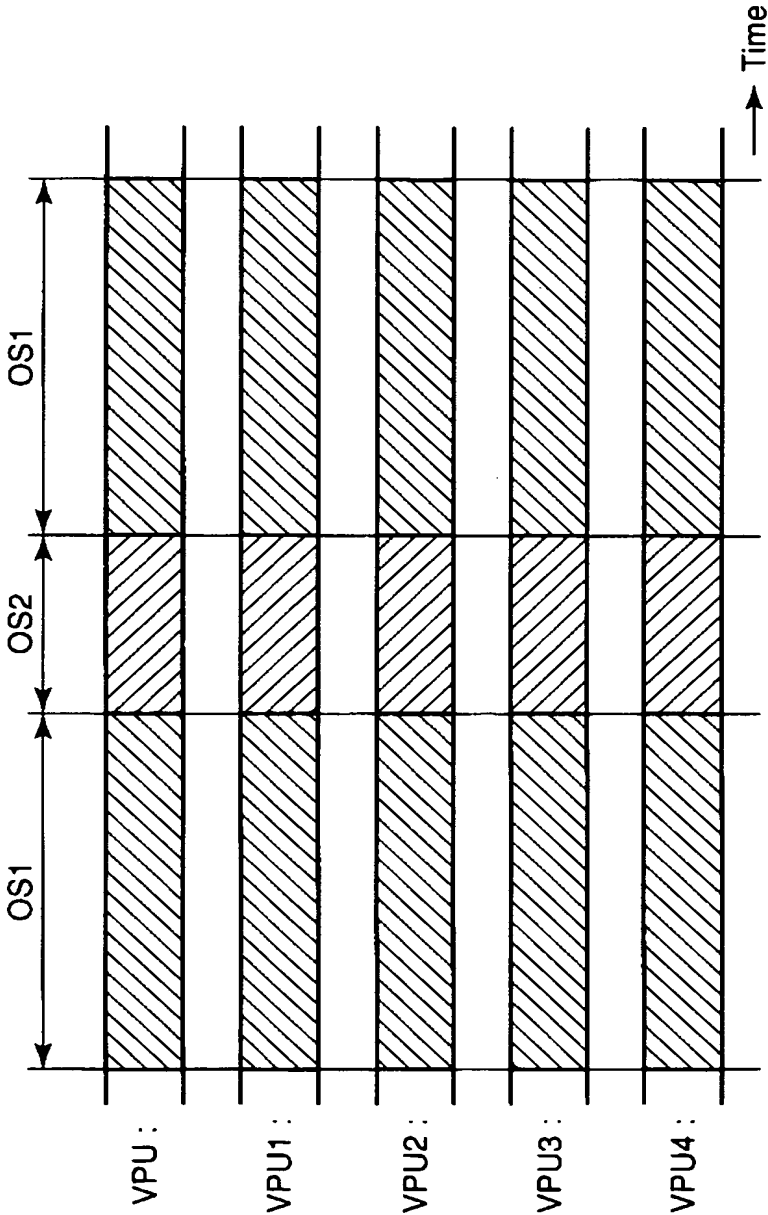
【図 13】



【図 14】

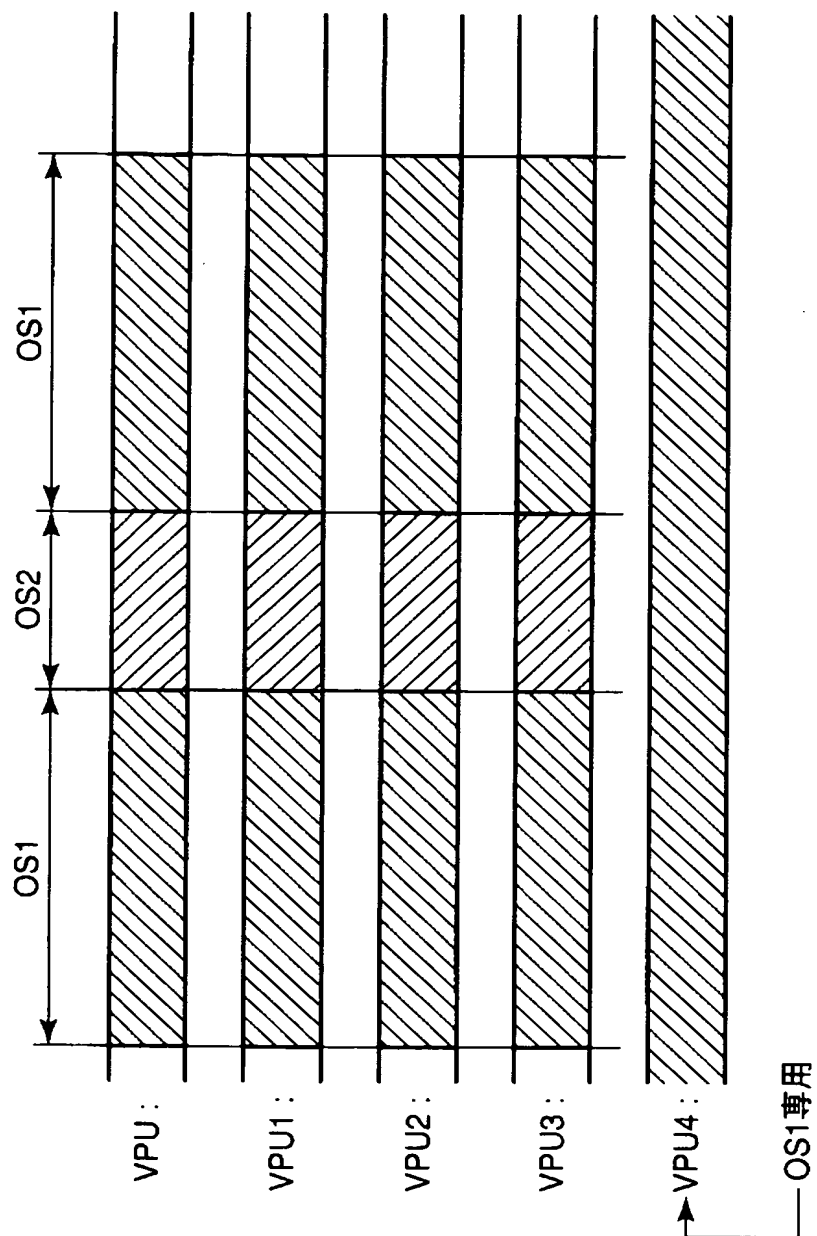


【図 15】

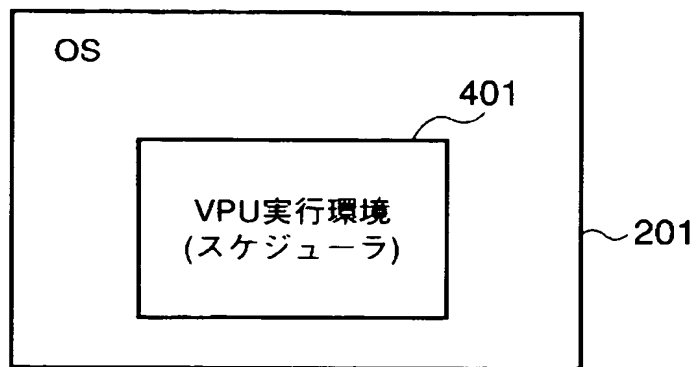




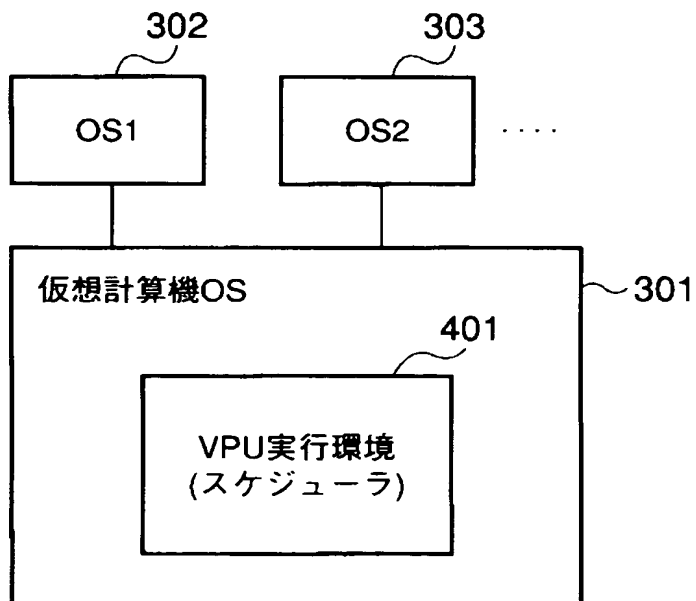
【図 1 6】



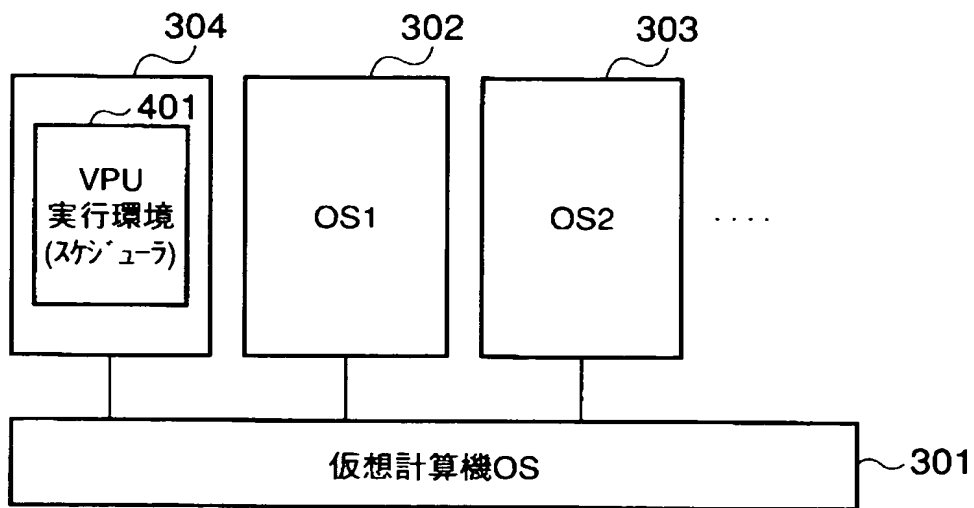
【図 17】



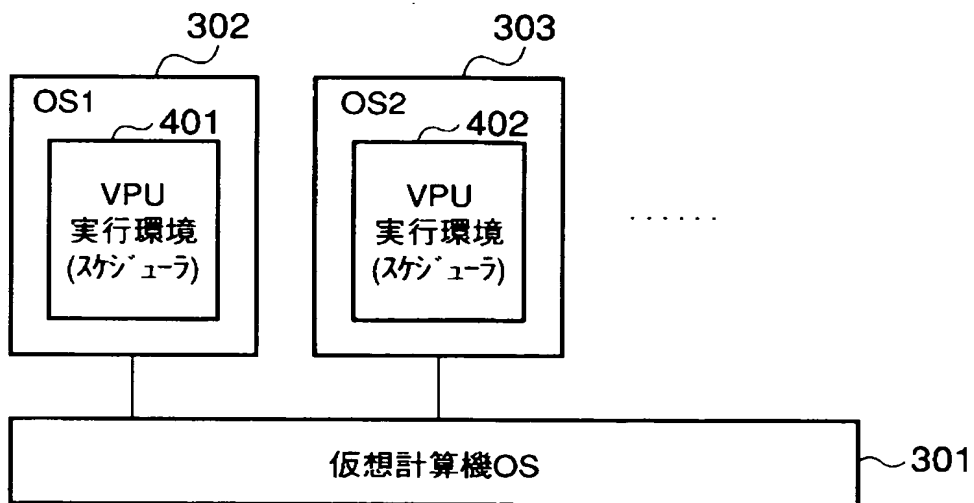
【図 18】



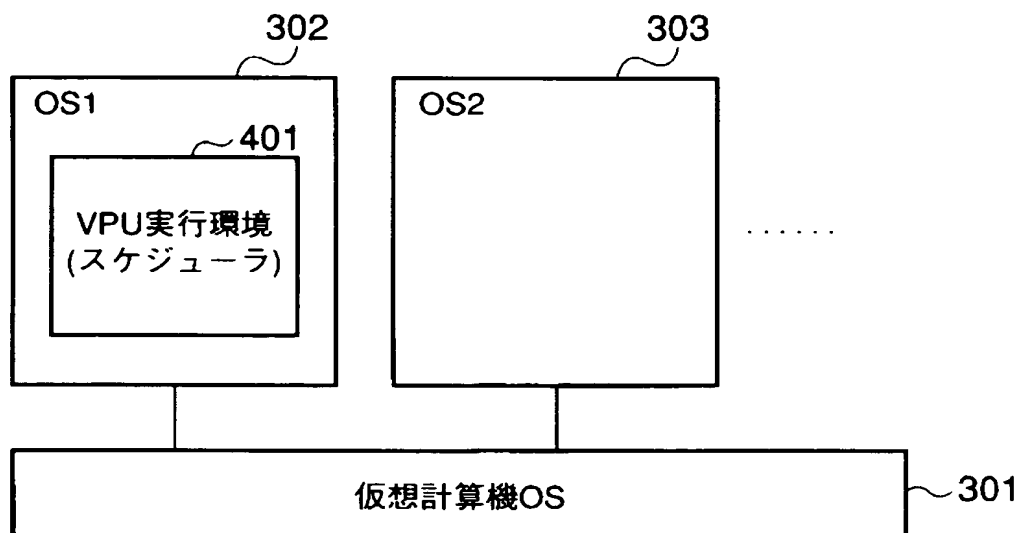
【図 19】



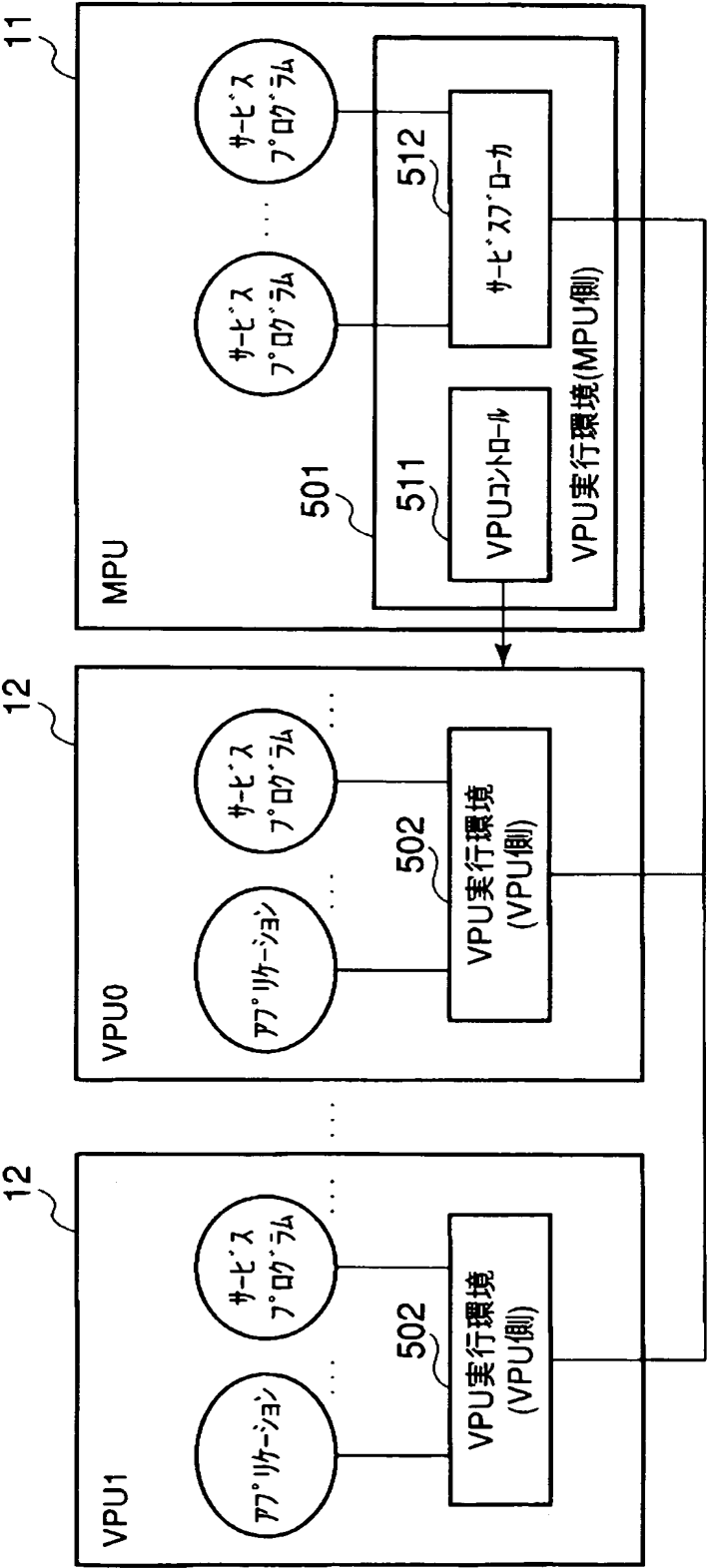
【図 20】



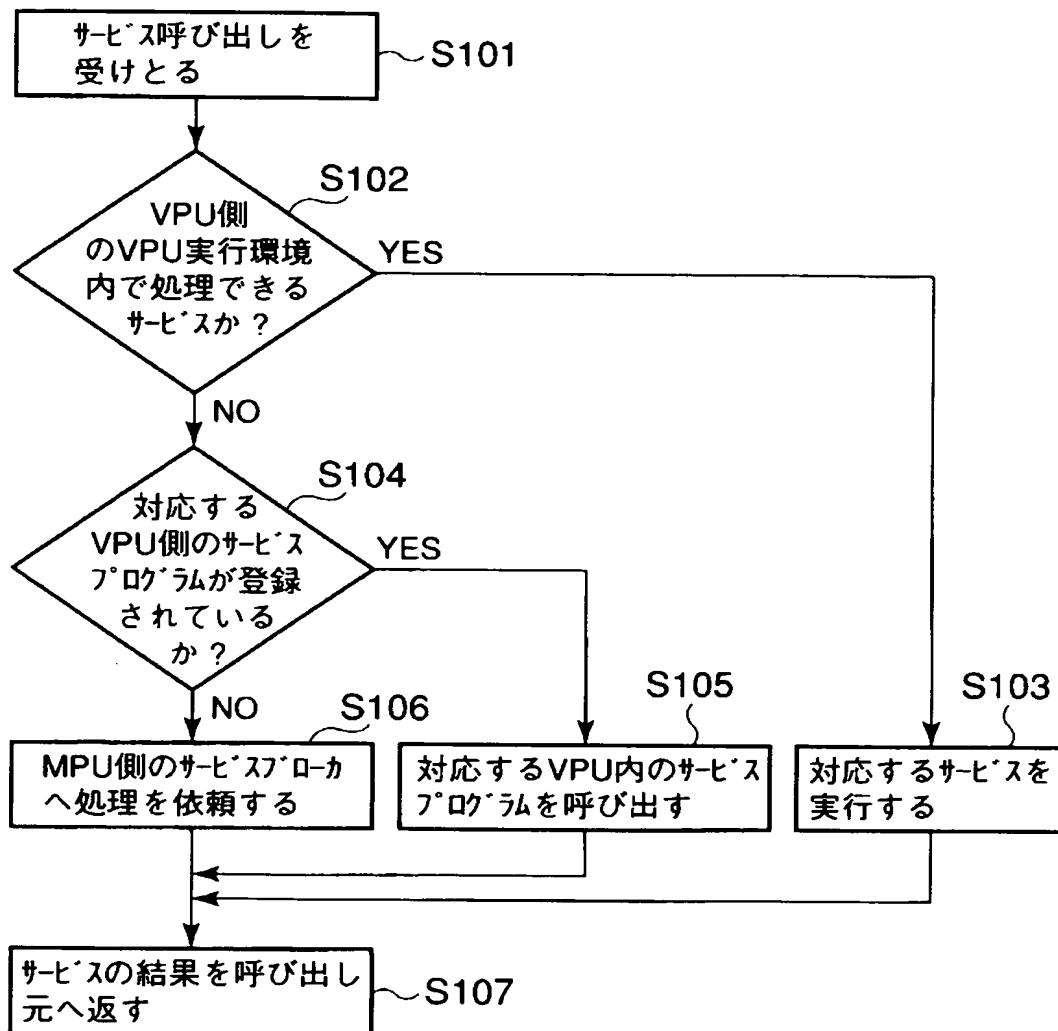
【図 21】



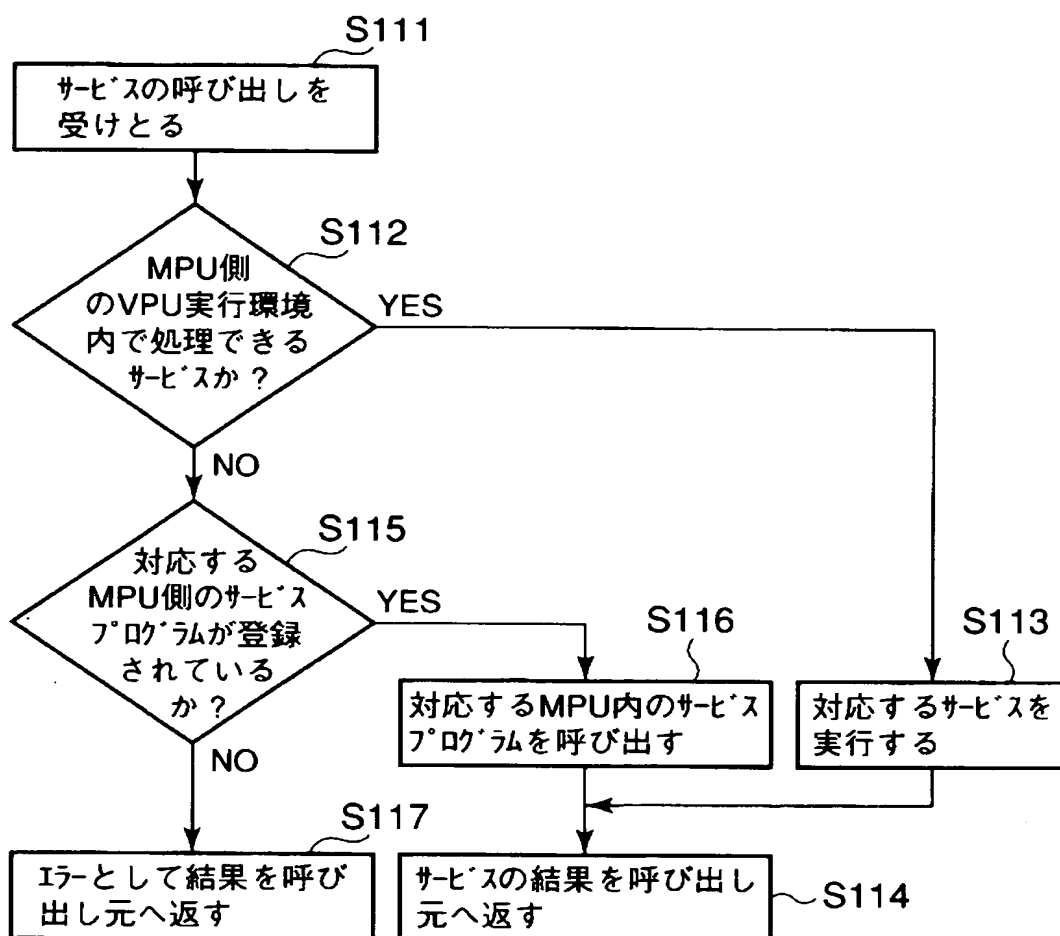
【図 22】



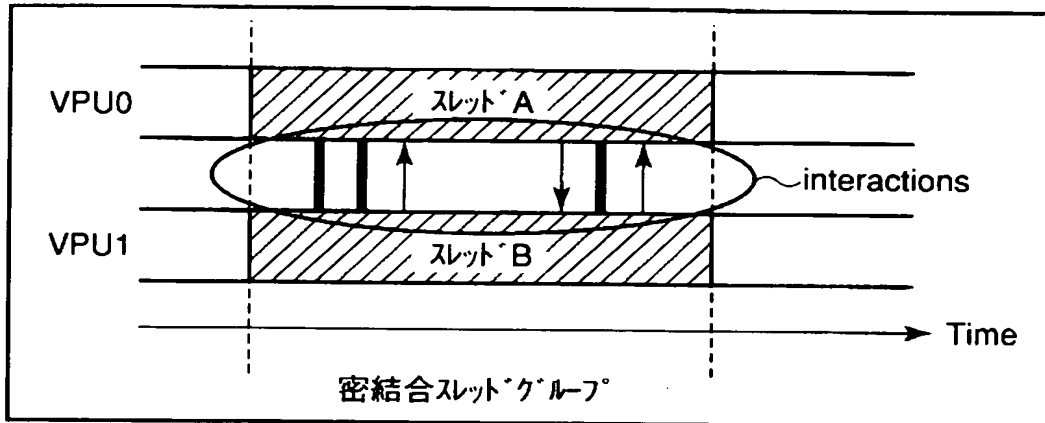
【図 23】



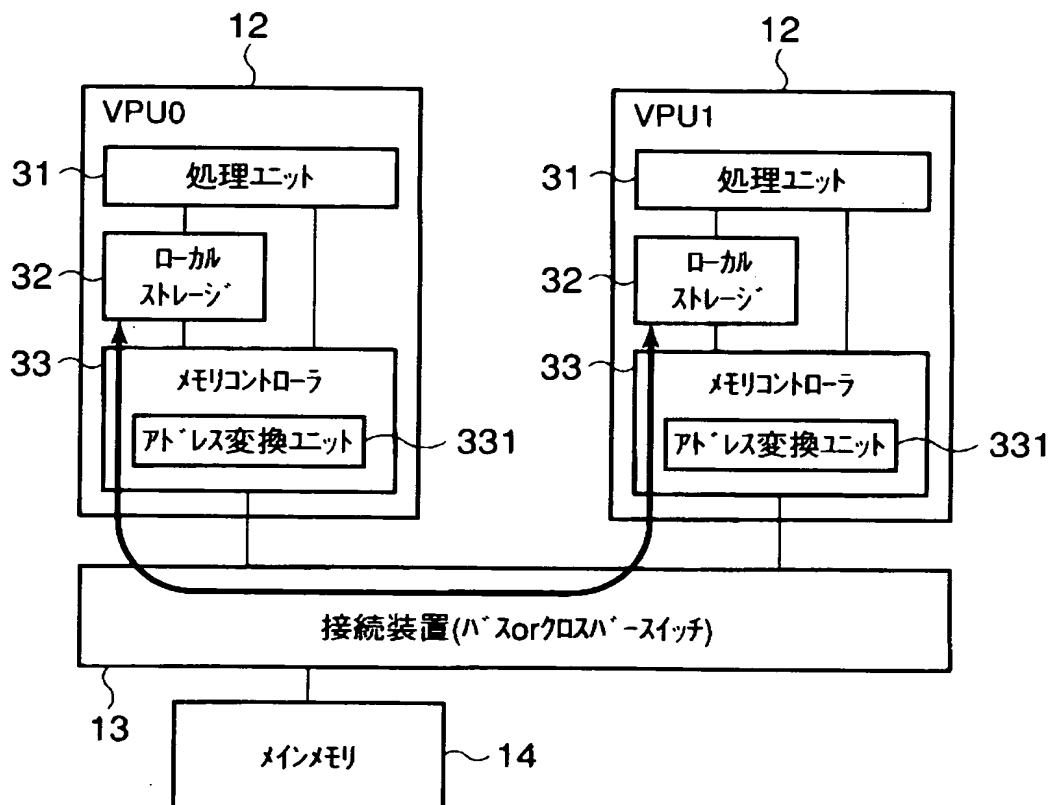
【図 24】



【図 25】

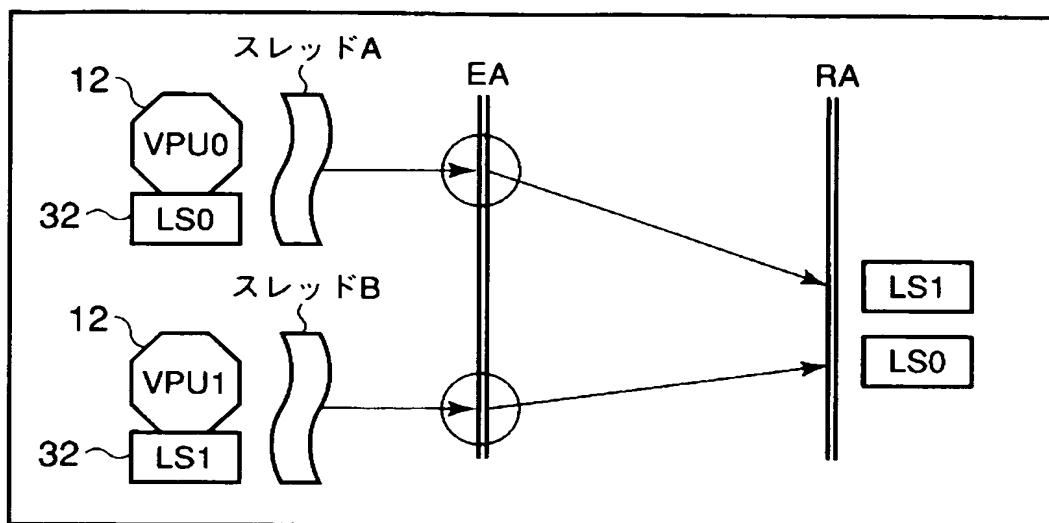


【図 26】

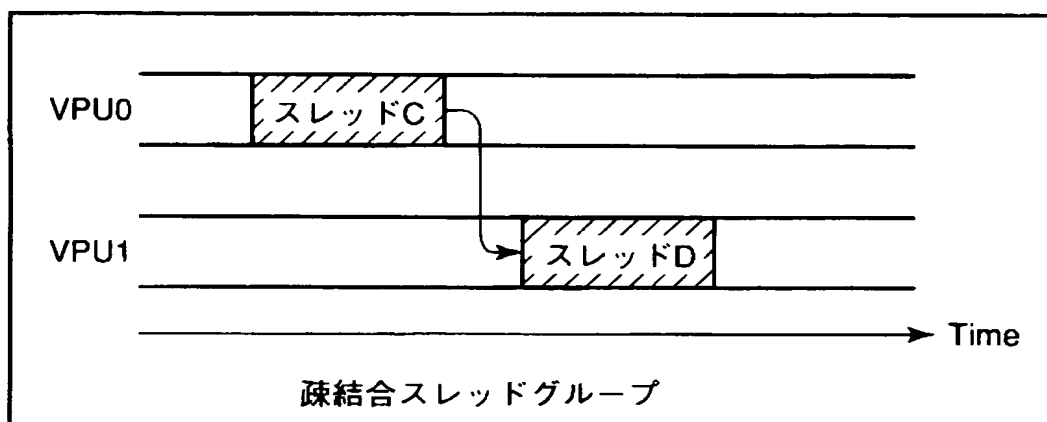




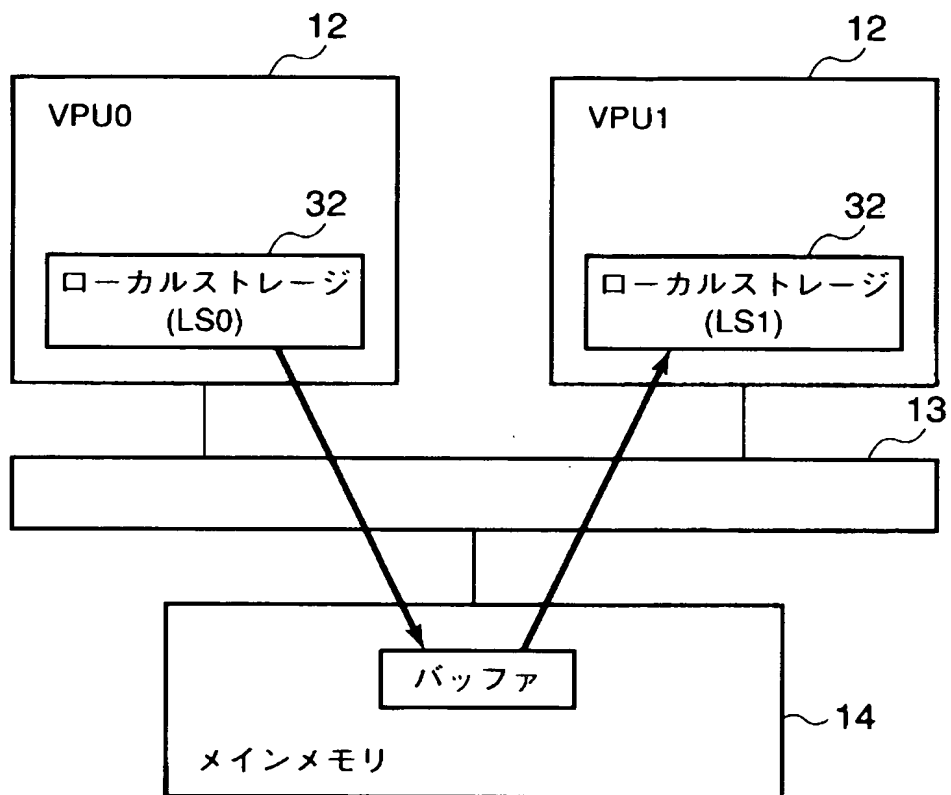
【図 27】



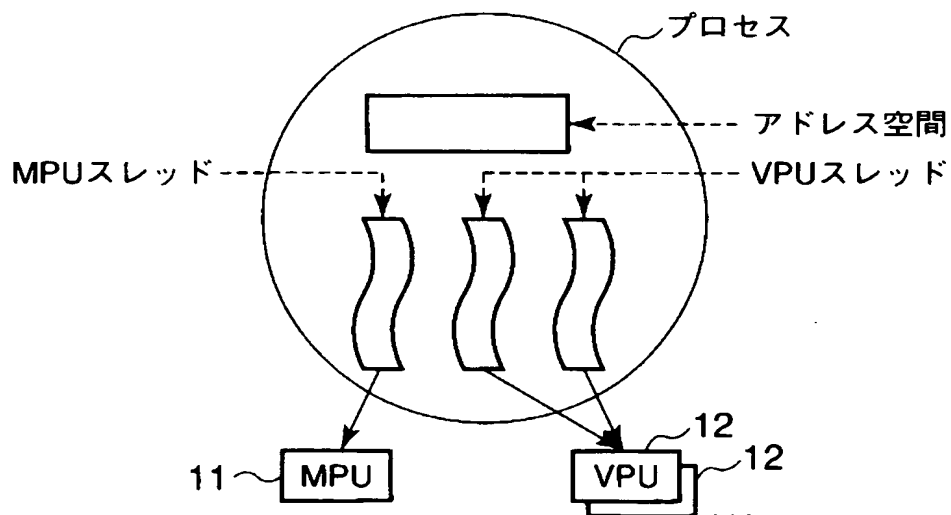
【図 28】



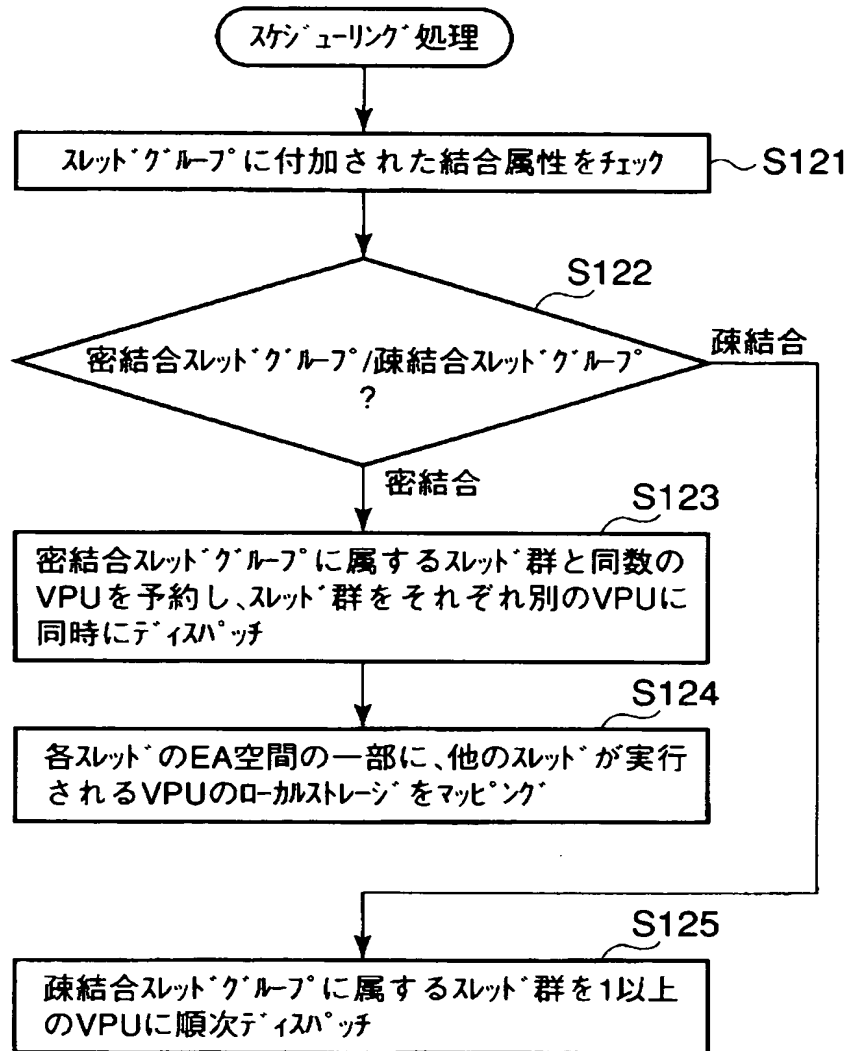
【図 29】



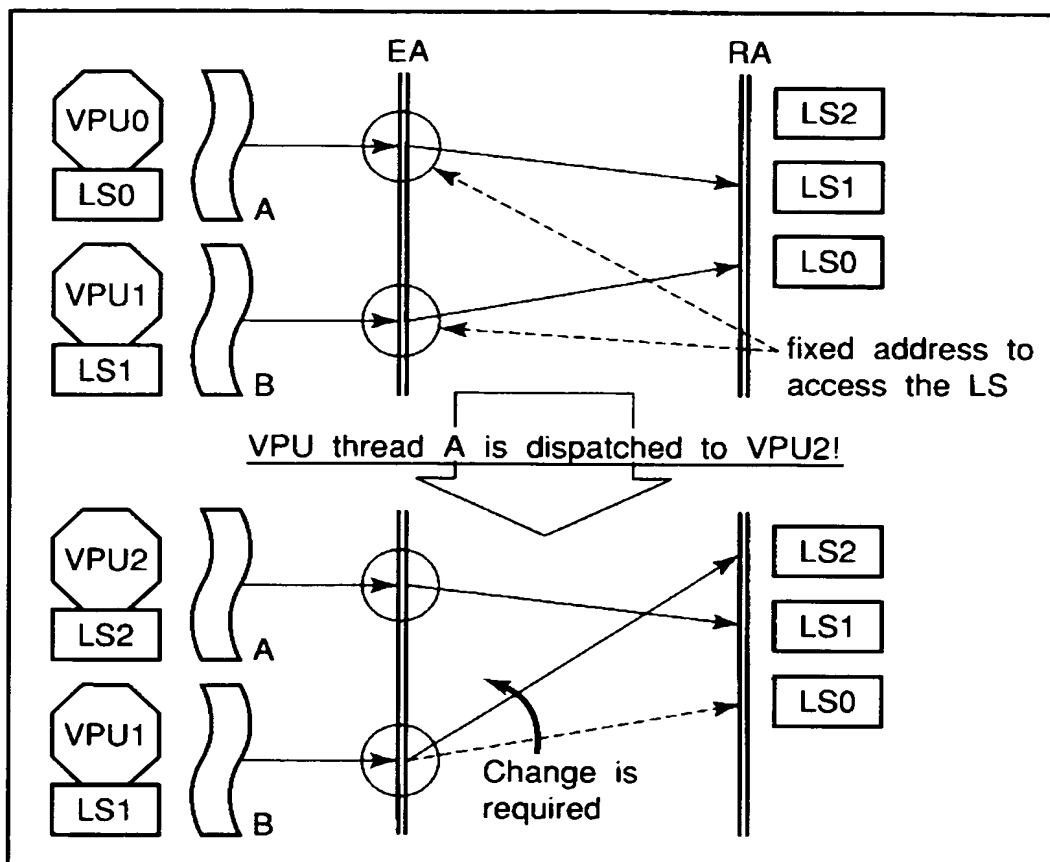
【図 30】



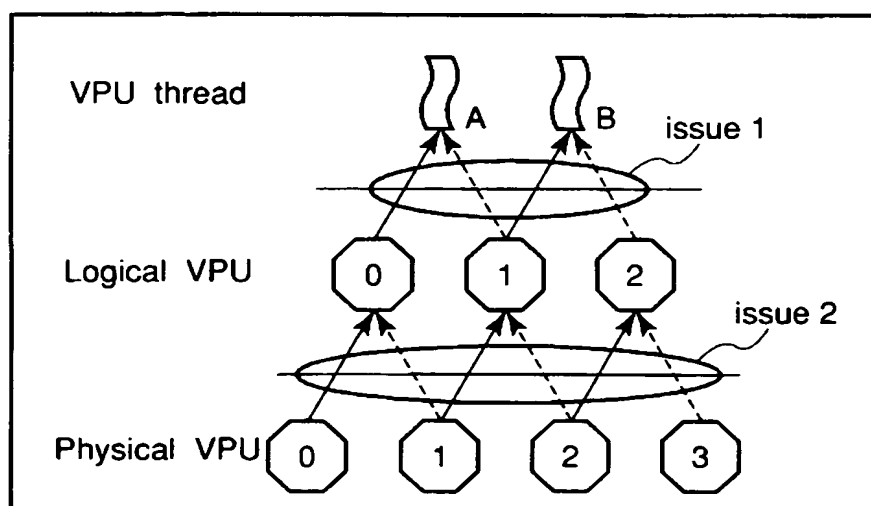
【図 31】



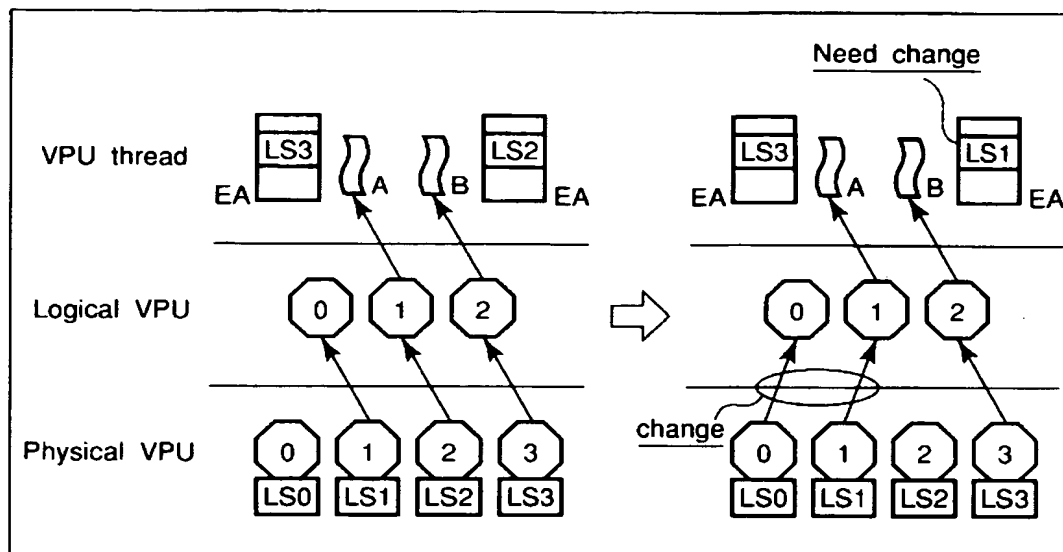
【図 3 2】



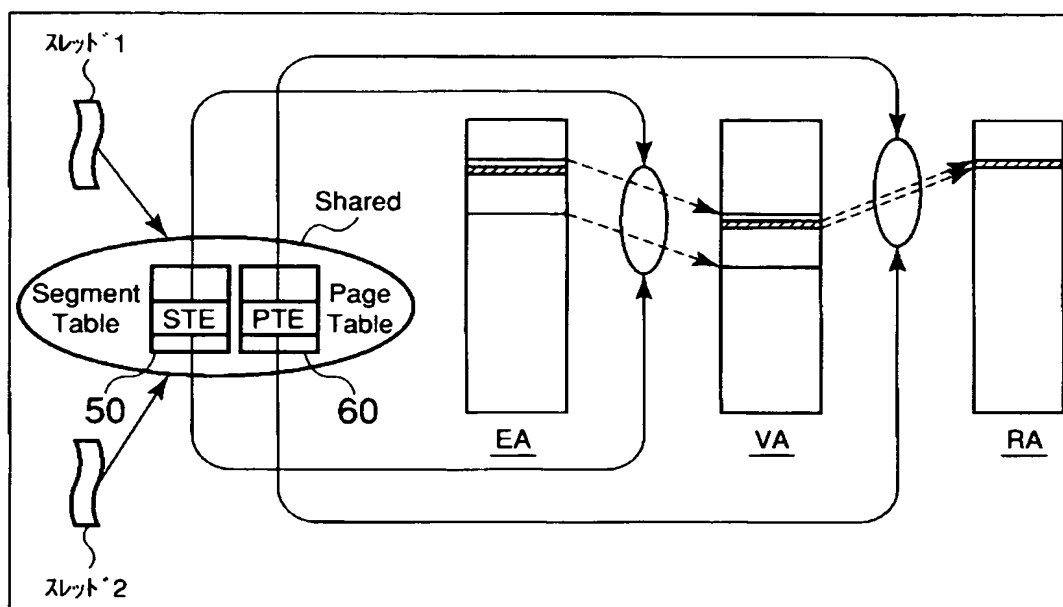
【図 3 3】



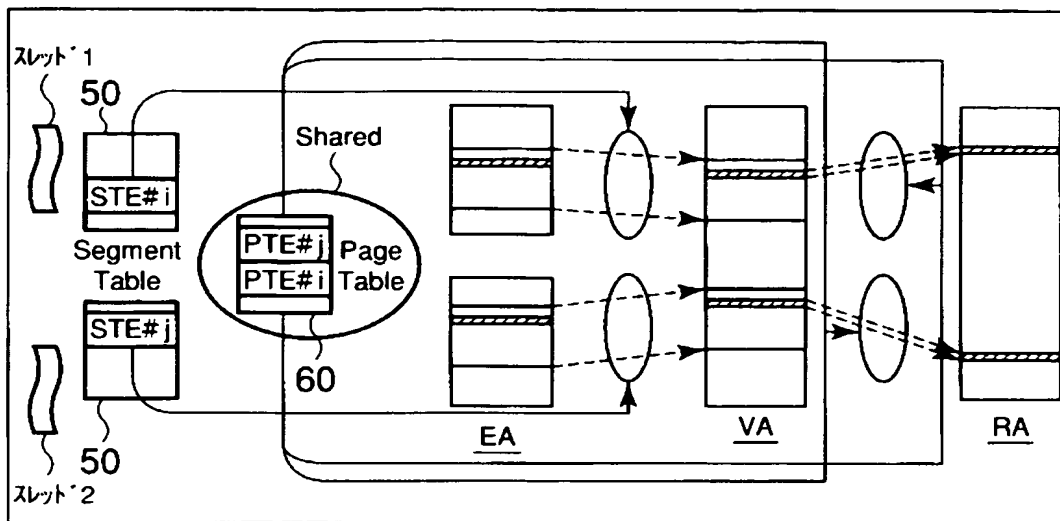
【図 3 4】



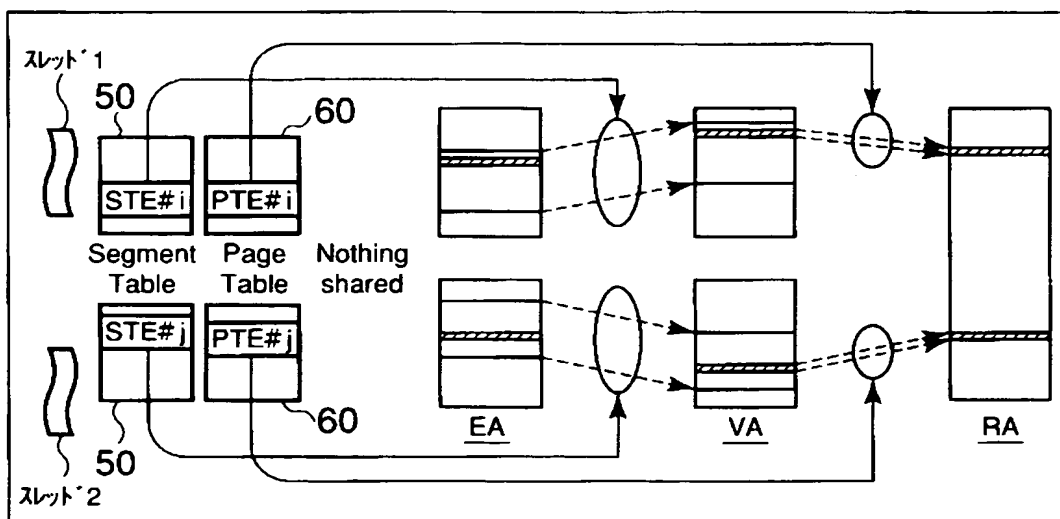
【図 3 5】



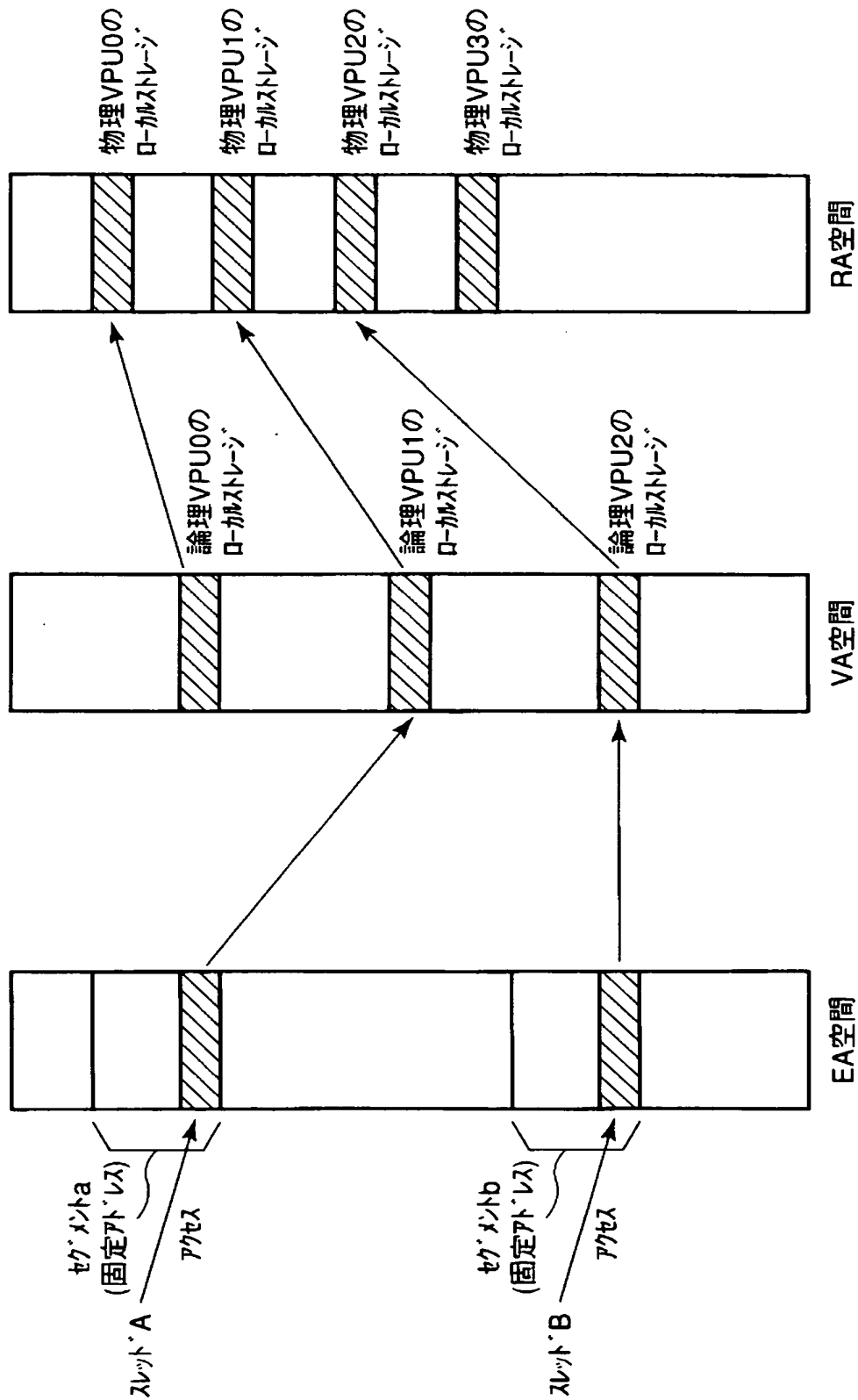
【図 3 6】



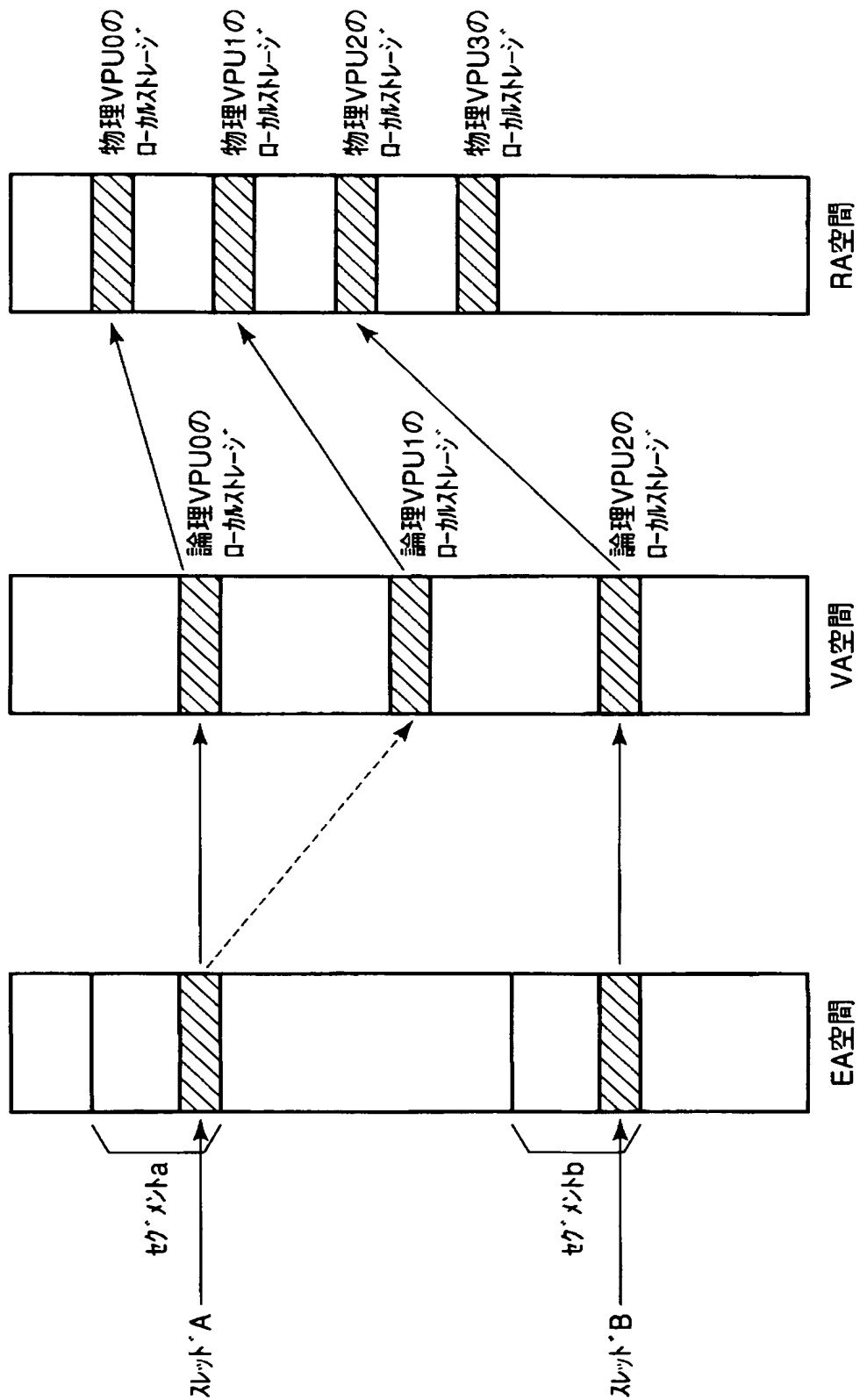
【図 3 7】



【図 38】

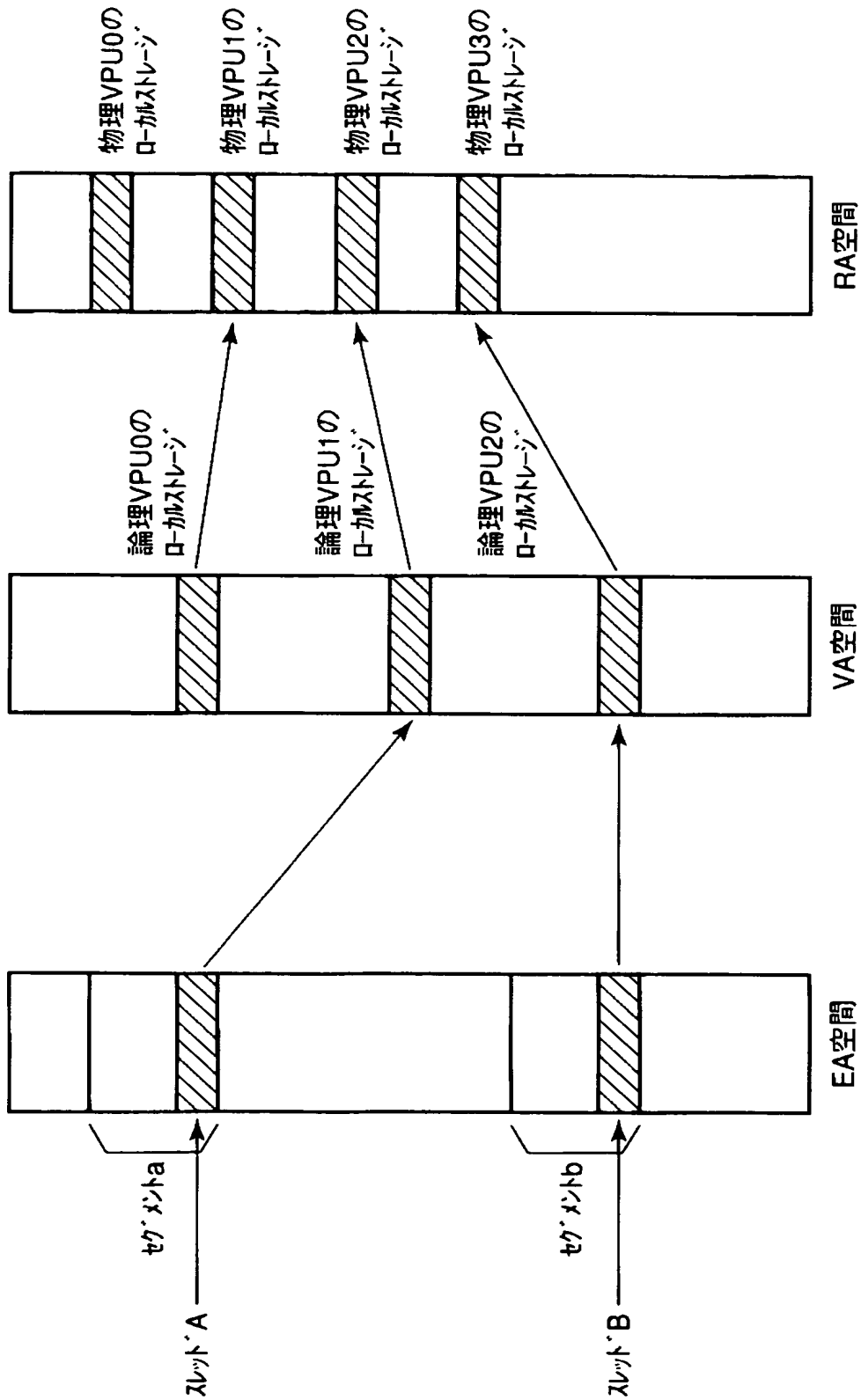


【図 39】

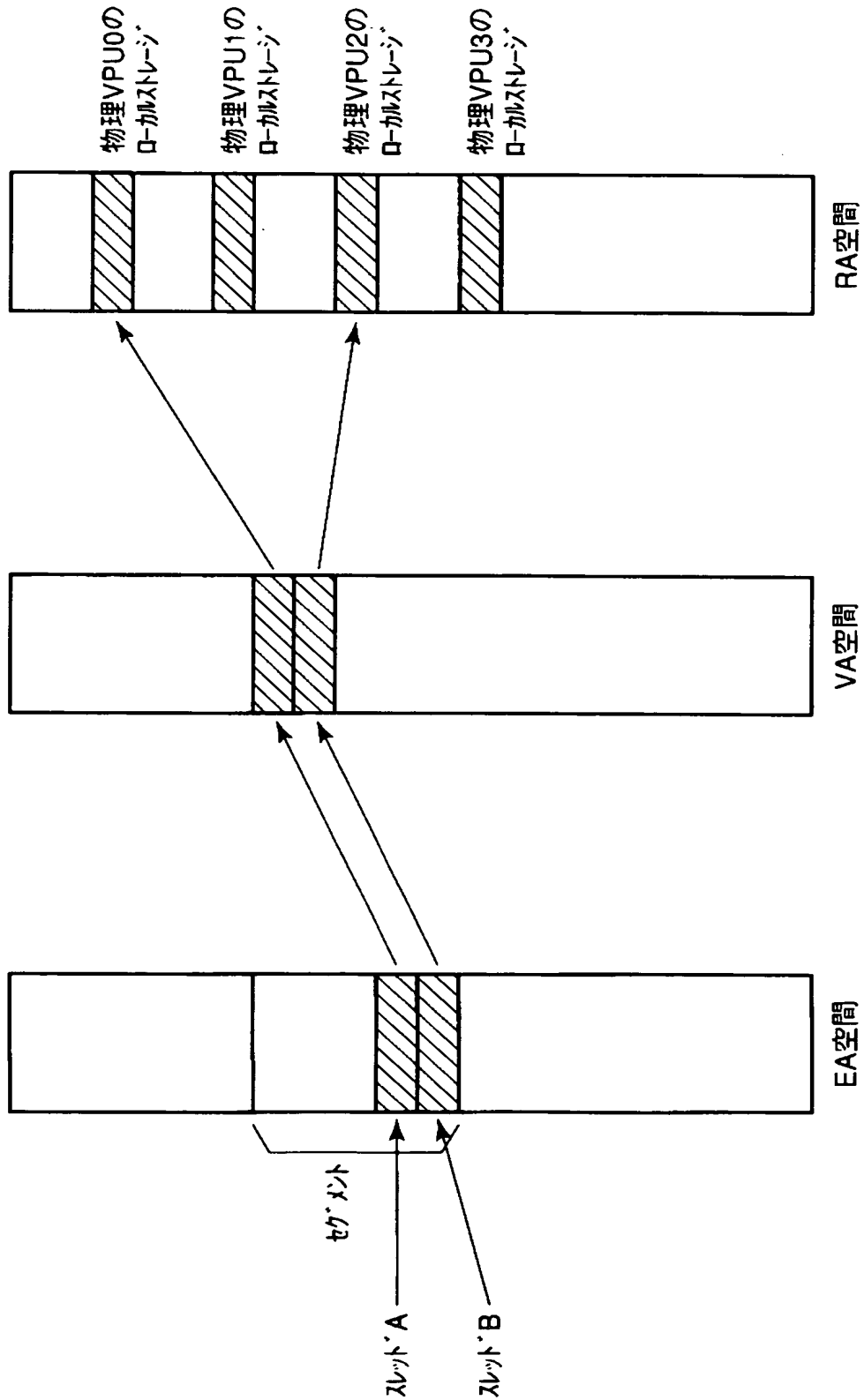




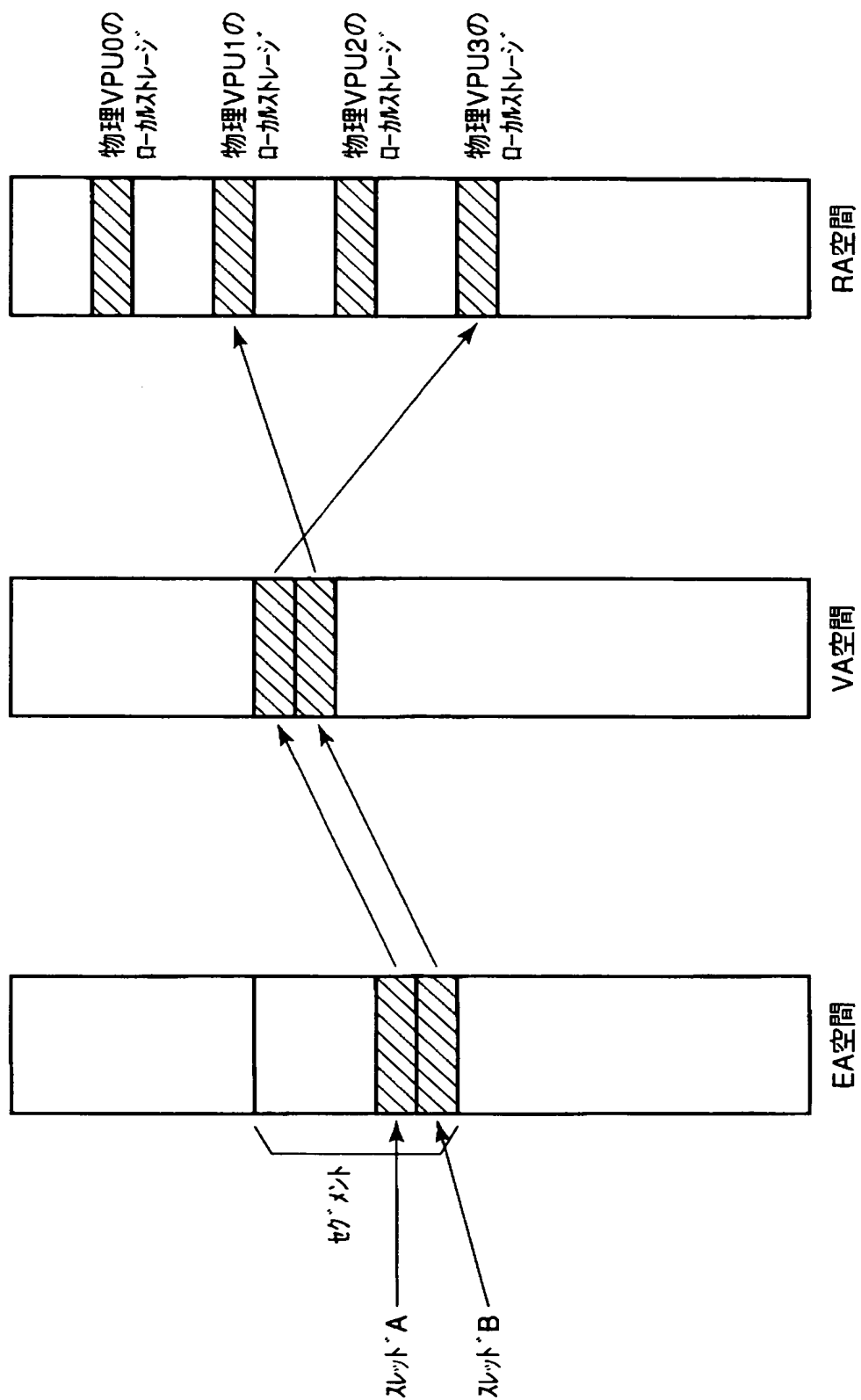
【図 40】



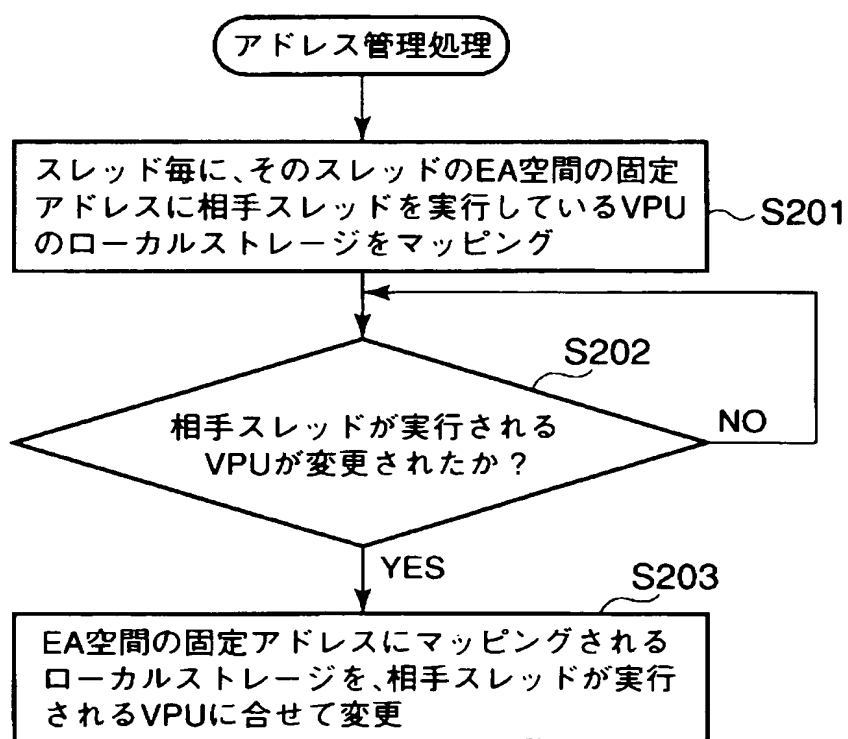
【図 4 1】



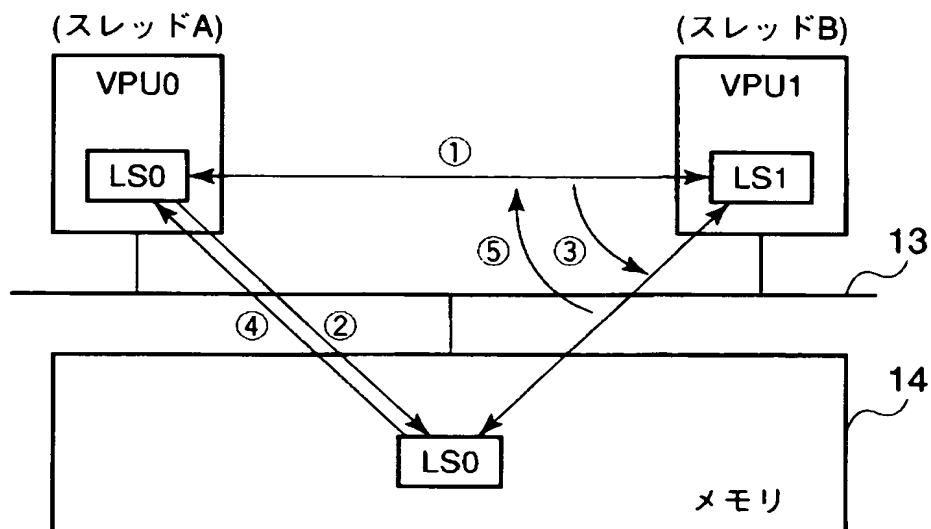
【図 4 2】



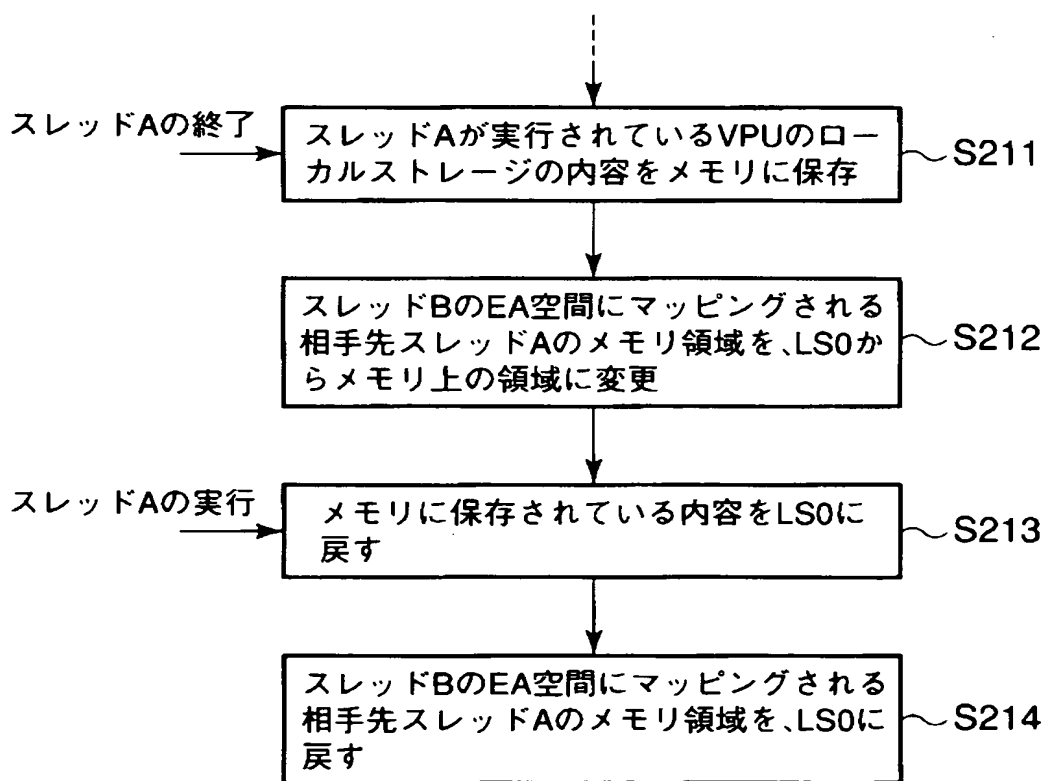
【図 43】



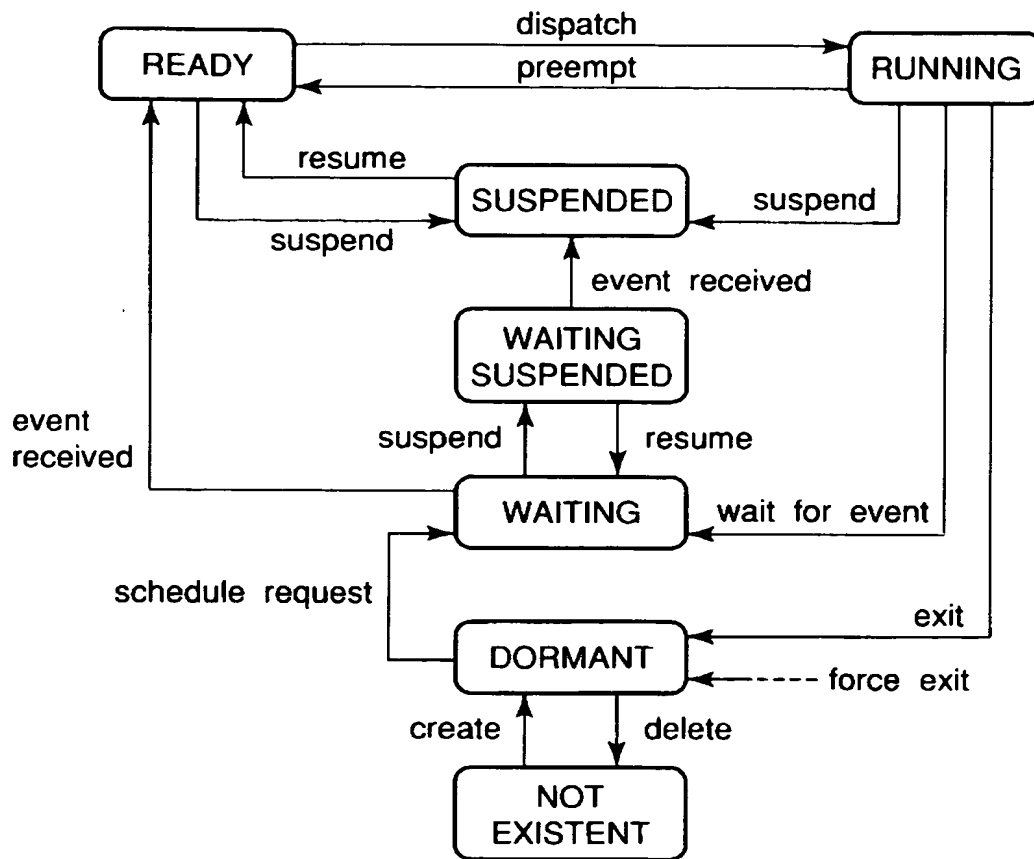
【図 4 4】



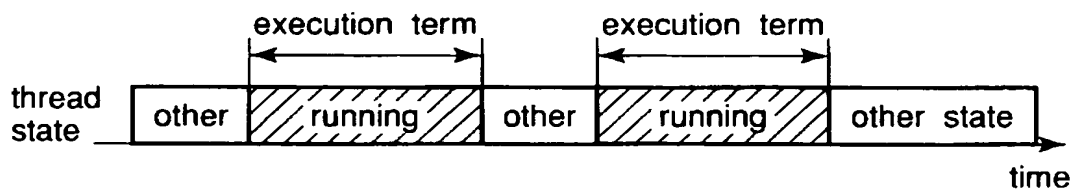
【図 4 5】



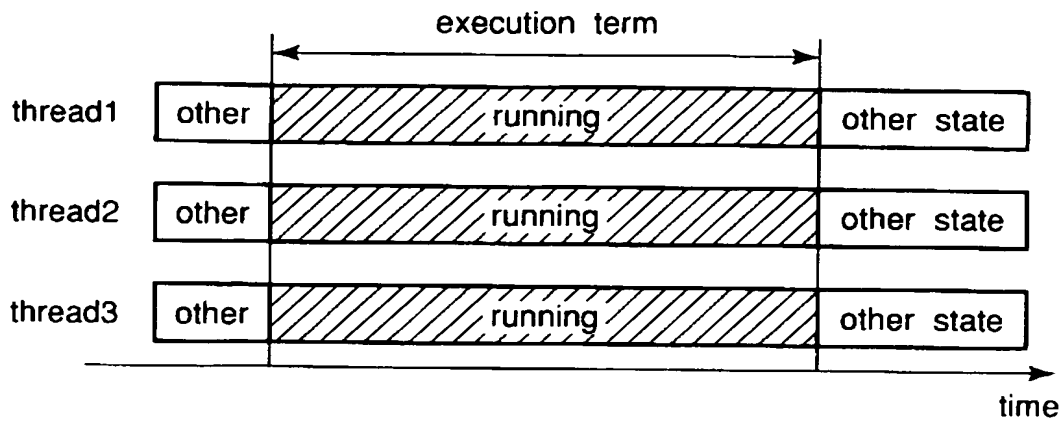
【図 4 6】



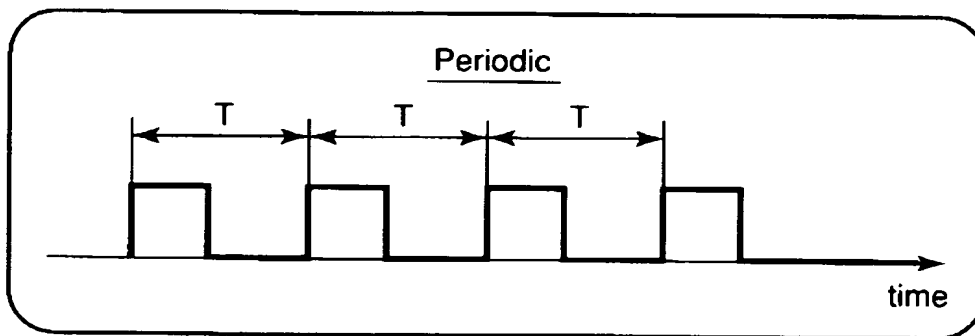
【図 4 7】



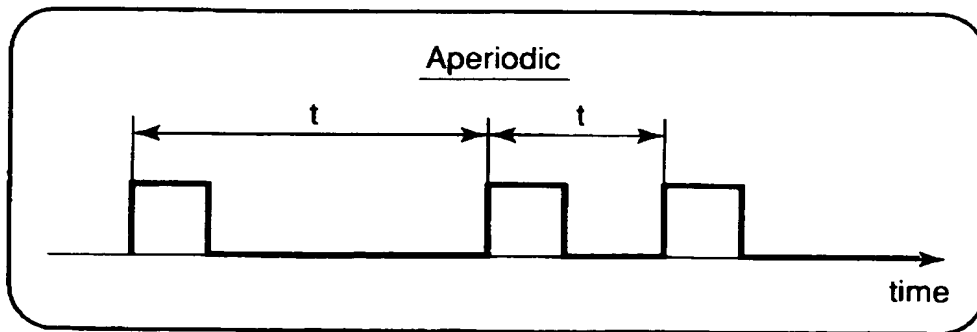
【図 48】



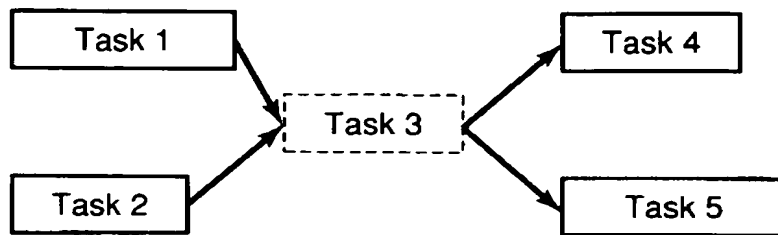
【図 49】



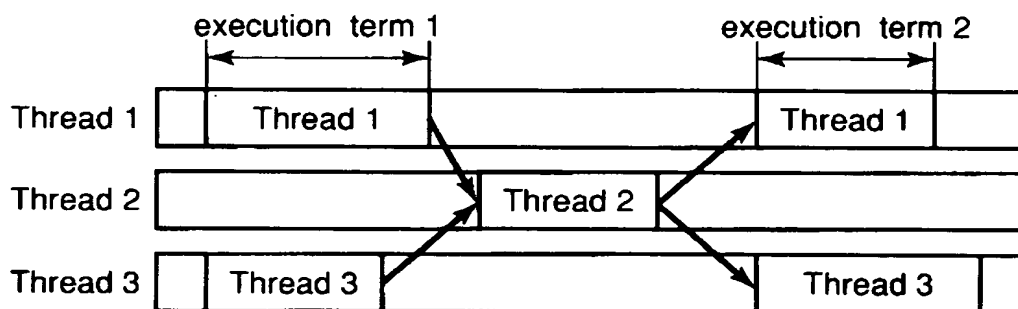
【図 50】



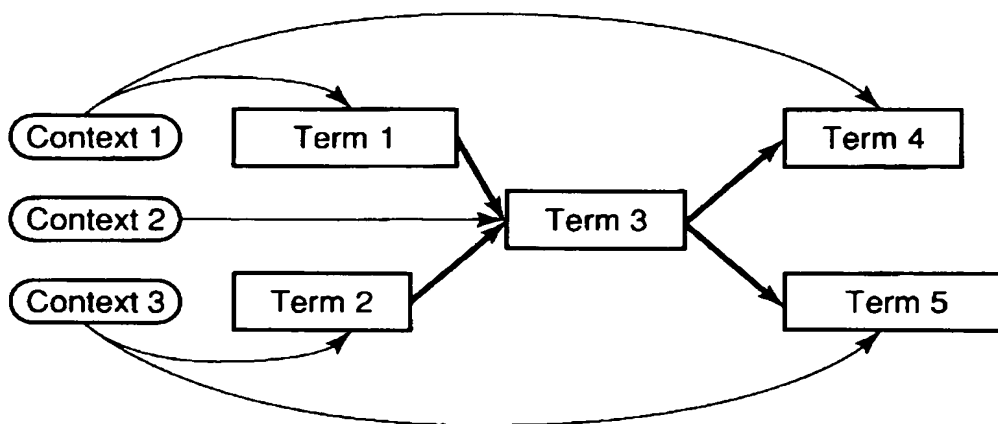
【図 5 1】



【図 5 2】

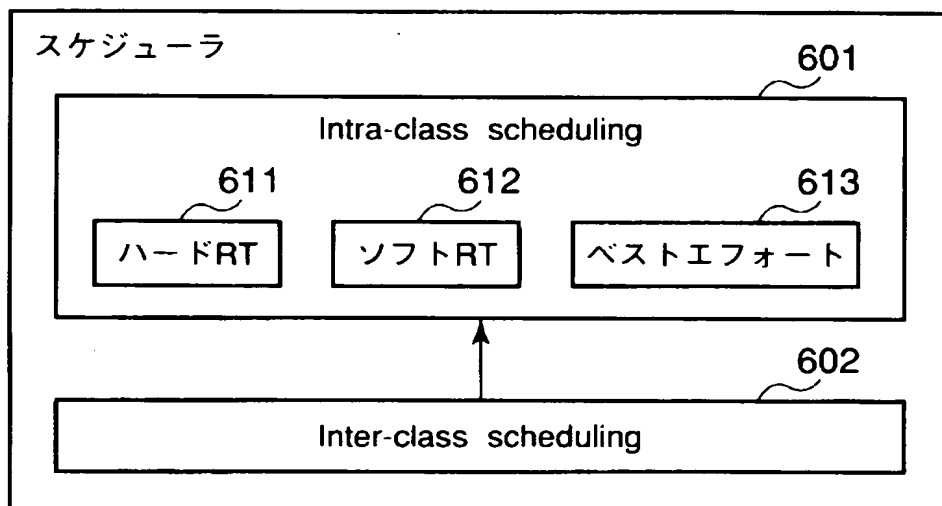


【図 5 3】

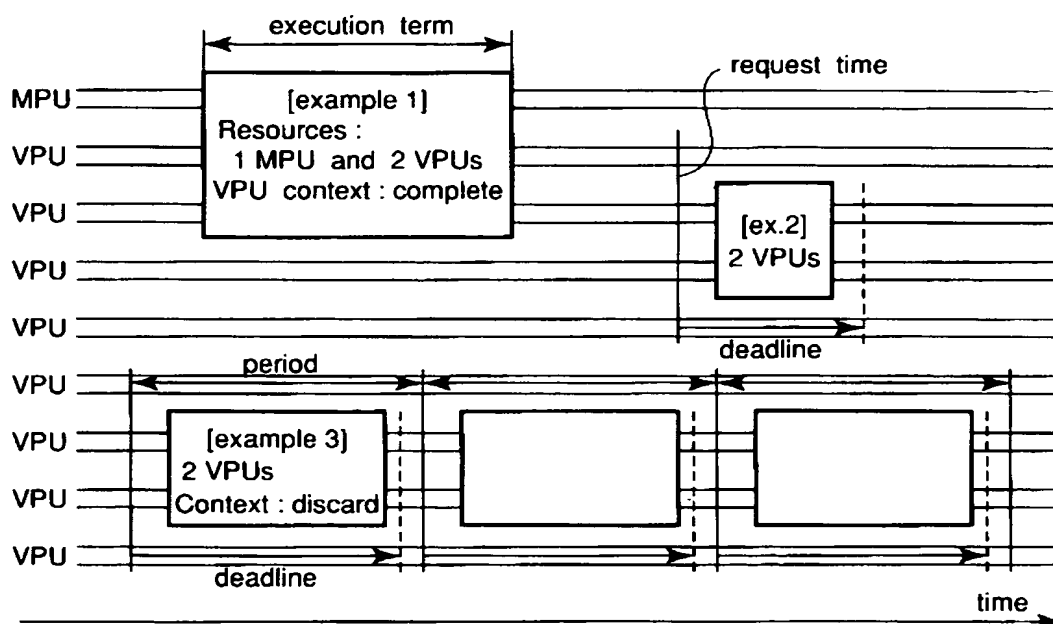




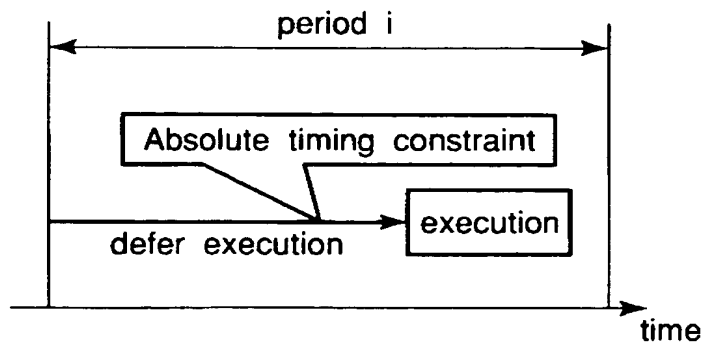
【図 5 4】



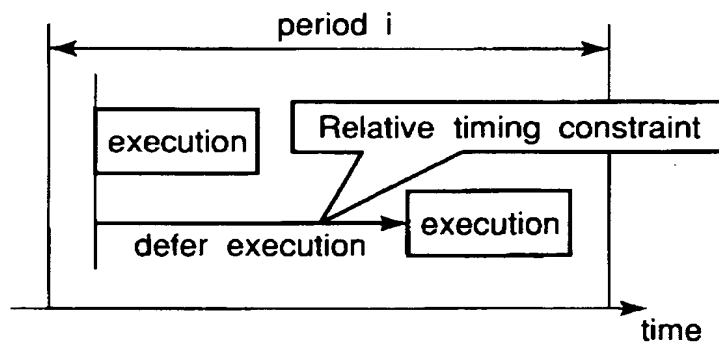
【図 5 5】



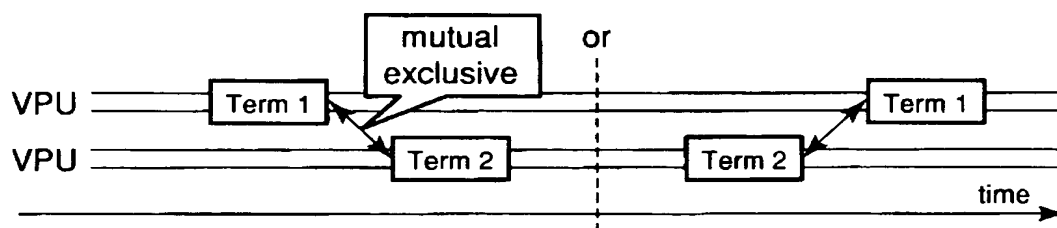
【図 5 6】



【図 5 7】



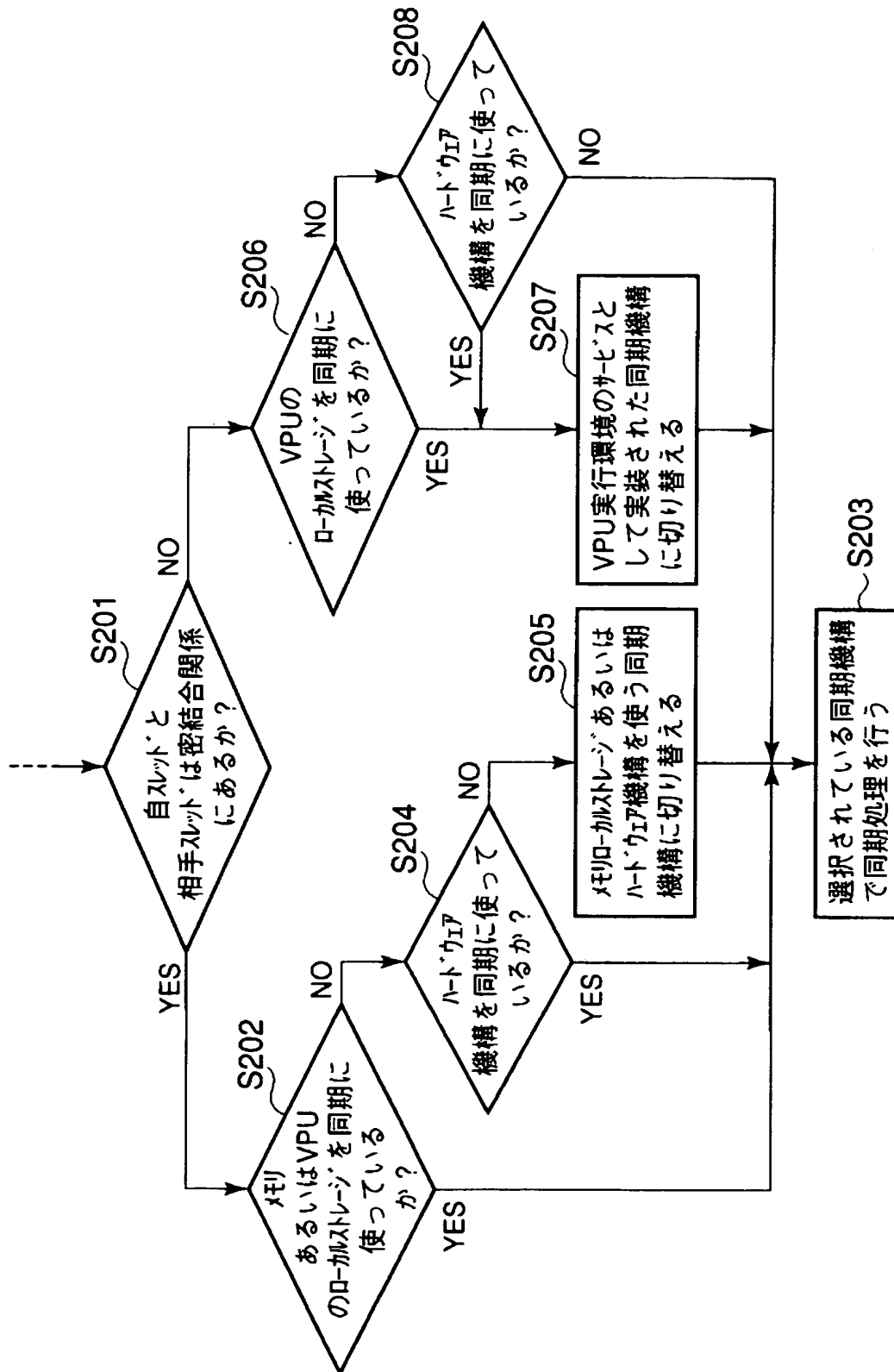
【図 5 8】



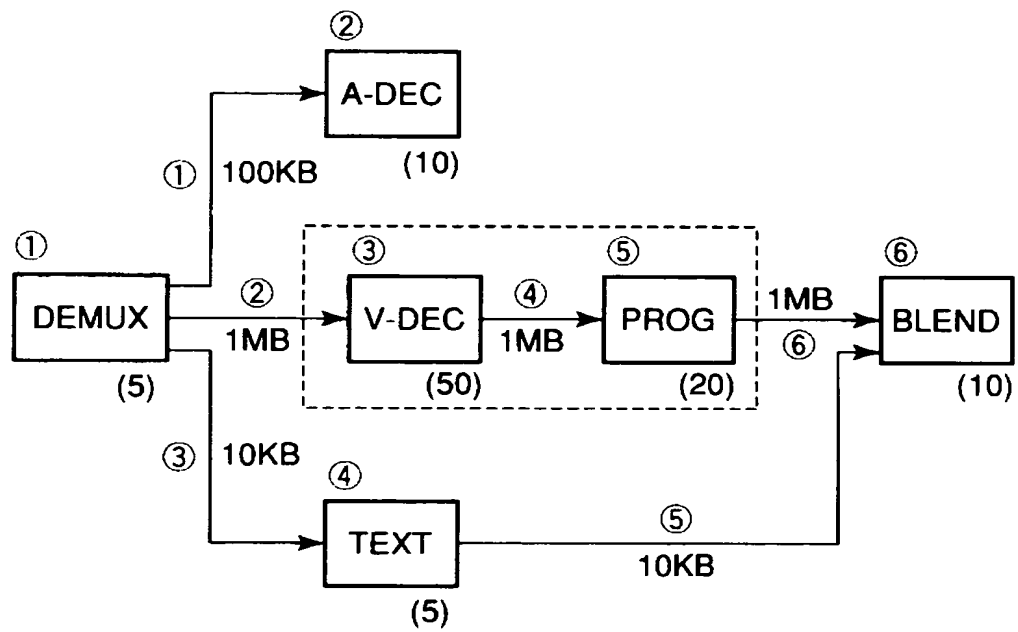
【図 5 9】

		Tightly coupled thread group	Loosely coupled thread group
On memory	LS	Can use	Cannot use
	MS	Can use	
Other		Should use hardware primitives	Should use mechanisms provided by VPU Runtime Environment

【図 60】



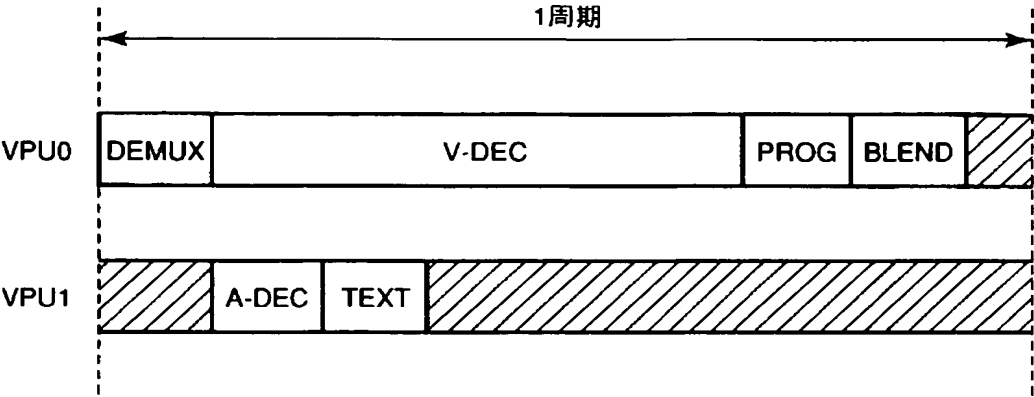
【図 61】



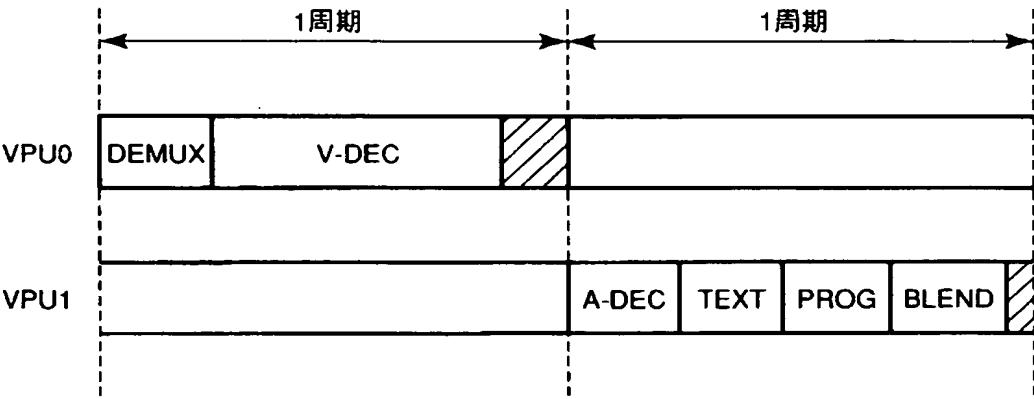
【図 6 2】

<u>BUFFER</u>	Id : 1
Size : 100KB	SrcTask : 1 DstTask : 2
<u>BUFFER</u>	Id : 2
Size : 1MB	SrcTask : 1 DstTask : 3
<u>BUFFER</u>	Id : 3
Size : 10KB	SrcTask : 1 DstTask : 4
<u>BUFFER</u>	Id : 4
Size : 1MB	SrcTask : 3 DstTask : 5
<u>BUFFER</u>	Id : 5
Size : 10KB	SrcTask : 4 DstTask : 6
<u>BUFFER</u>	Id : 6
Size : 1MB	SrcTask : 5 DstTask : 6
<u>TASK</u>	Id : 1 Class : VPU,HRT
ThreadContext : DEMUX	Cost : 5
Constraint : Precede : 2,3,4	
InputBuffer :	OutputBuffer : 1,2,3
<u>TASK</u>	Id : 2 Class : VPU,HRT
ThreadContext : A-DEC	Cost : 10
Constraint : Precede :	
InputBuffer : 1	OutputBuffer :
<u>TASK</u>	Id : 3 Class : VPU,HRT
ThreadContext : V-DEC	Cost : 50
Constraint : Precede : 5	
InputBuffer : 2	OutputBuffer : 4
<u>TASK</u>	Id : 4 Class : VPU,HRT
ThreadContext : TEXT	Cost : 5
Constraint : Precede : 6	
InputBuffer : 3	OutputBuffer : 5
<u>TASK</u>	Id : 5 Class : VPU,HRT
ThreadContext : PROG	Cost : 20
Constraint : Precede : 6	
InputBuffer : 4	OutputBuffer : 6
<u>TASK</u>	Id : 6 Class : VPU,HRT
ThreadContext : BLEND	Cost : 10
Constraint : Precede : 5	
InputBuffer : 5,6	OutputBuffer :

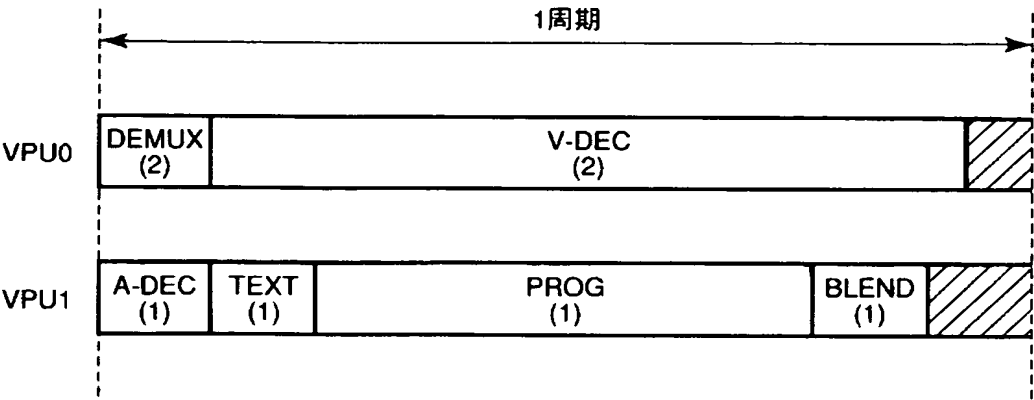
【図 6 3】



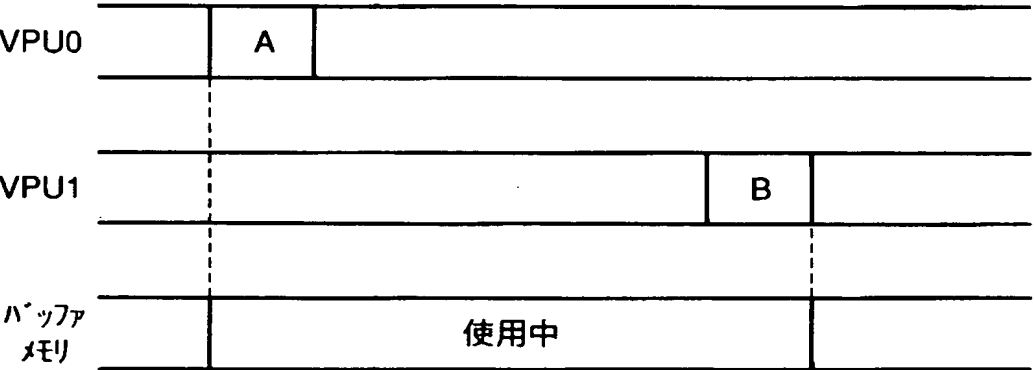
【図 6 4】



【図 6 5】

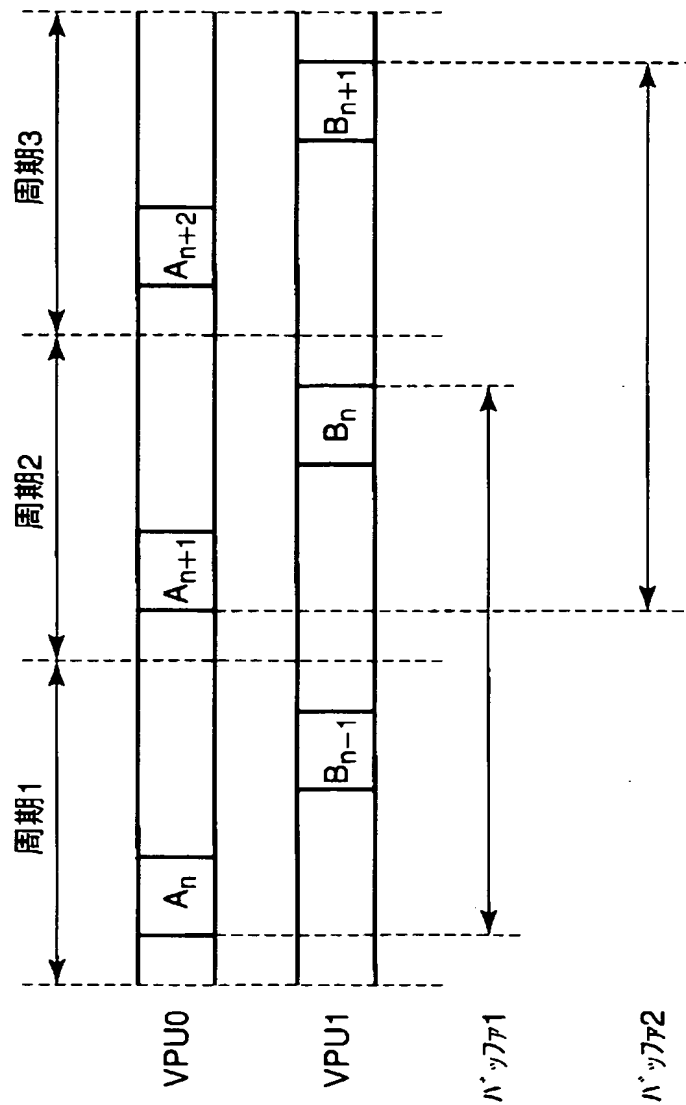


【図 6 6】

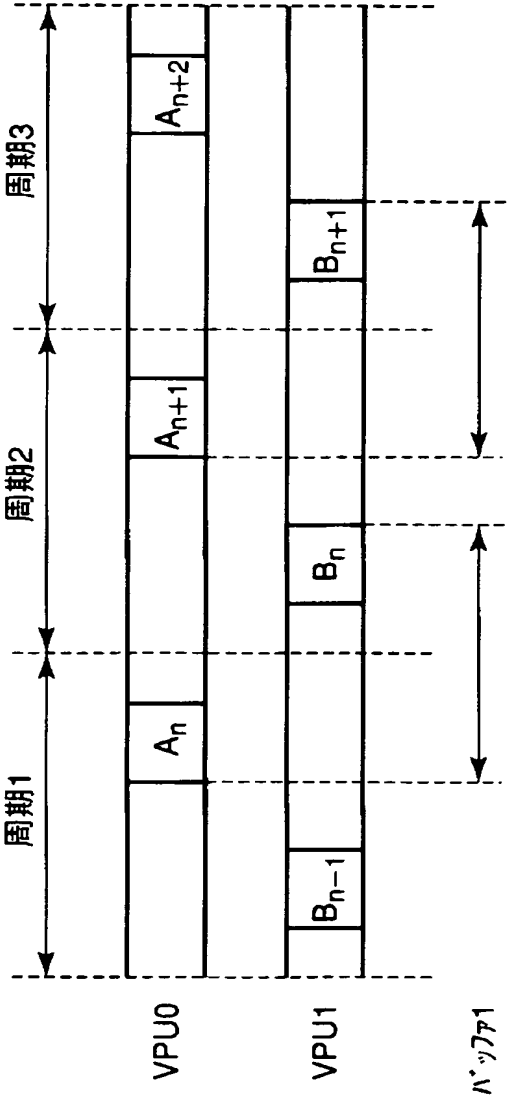




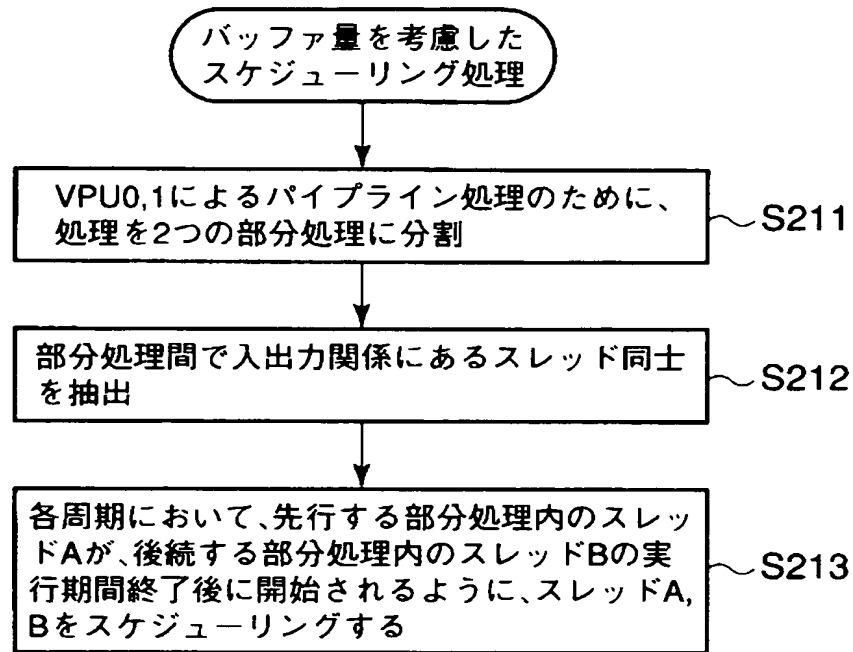
【図 67】



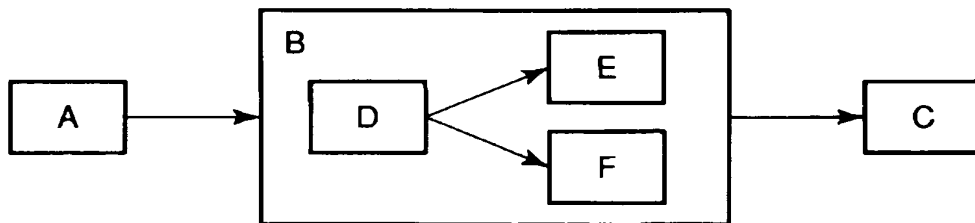
【図 68】



【図 69】



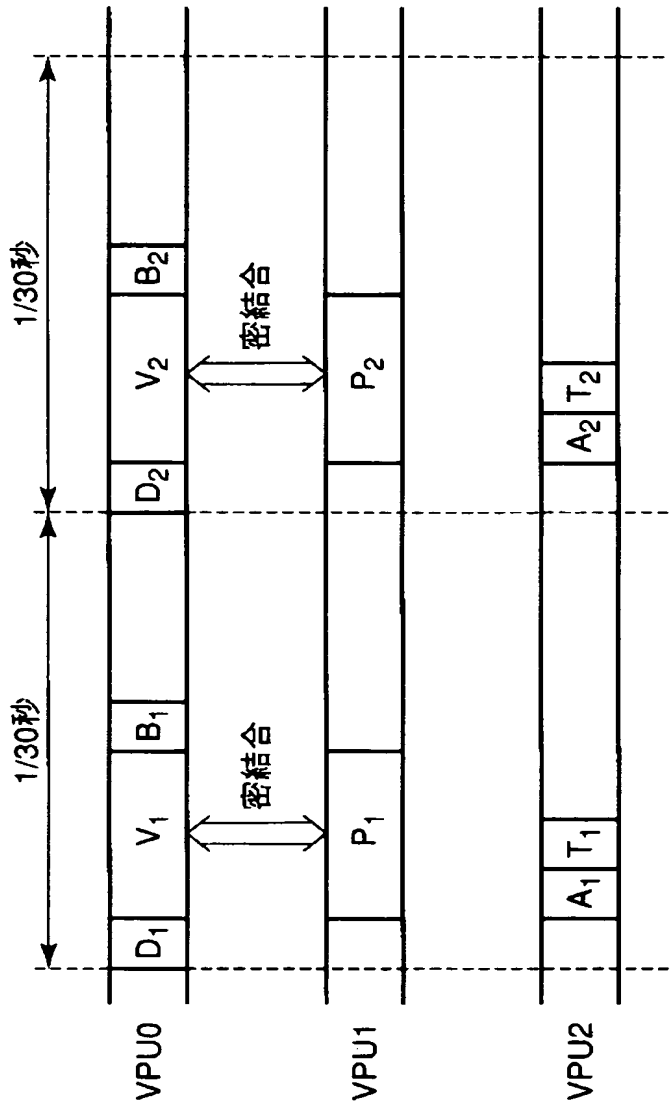
【図 70】



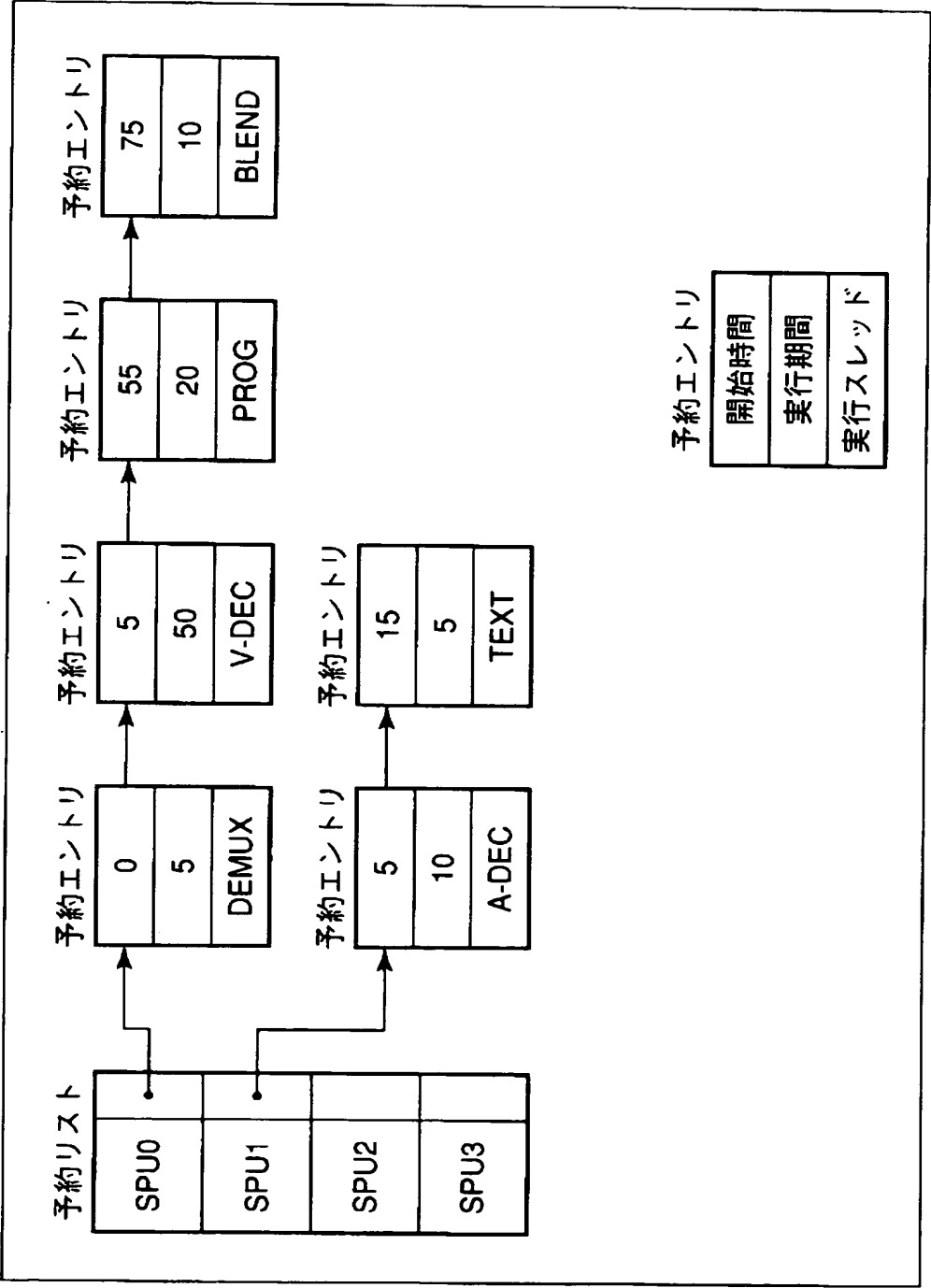
【図 71】

<u>BUFFER</u>	Id : 1
Size : 100KB	SrcTask : 1 DstTask : 2
<u>BUFFER</u>	Id : 2
Size : 1MB	SrcTask : 1 DstTask : 3
<u>BUFFER</u>	Id : 3
Size : 10KB	SrcTask : 1 DstTask : 4
<u>BUFFER</u>	Id : 4
Size : 10KB	SrcTask : 4 DstTask : 6
<u>BUFFER</u>	Id : 5
Size : 1MB	SrcTask : 5 DstTask : 6
<u>TASK</u>	Id : 1 Class : VPU,HRT
ThreadContext : DEMUX	Cost : 5
Constraint : Precede : 2,3,4	
InputBuffer :	OutputBuffer : 1,2,3
<u>TASK</u>	Id : 2 Class : VPU,HRT
ThreadContext : A-DEC	Cost : 10
Constraint : Precede :	
InputBuffer : 1	OutputBuffer :
<u>TASK</u>	Id : 3 Class : VPU,HRT
ThreadContext : V-DEC	Cost : 50
Constraint : Precede : 5	TightlyCoupled : 5
InputBuffer : 2	OutputBuffer :
<u>TASK</u>	Id : 4 Class : VPU,HRT
ThreadContext : TEXT	Cost : 5
Constraint : Precede : 6	
InputBuffer : 3	OutputBuffer : 4
<u>TASK</u>	Id : 5 Class : VPU,HRT
ThreadContext : PROG	Cost : 50
Constraint : Precede : 6	TightlyCoupled : 3
InputBuffer :	OutputBuffer : 5
<u>TASK</u>	Id : 6 Class : VPU,HRT
ThreadContext : BLEND	Cost : 10
Constraint : Precede : 5	
InputBuffer : 4,5	OutputBuffer :

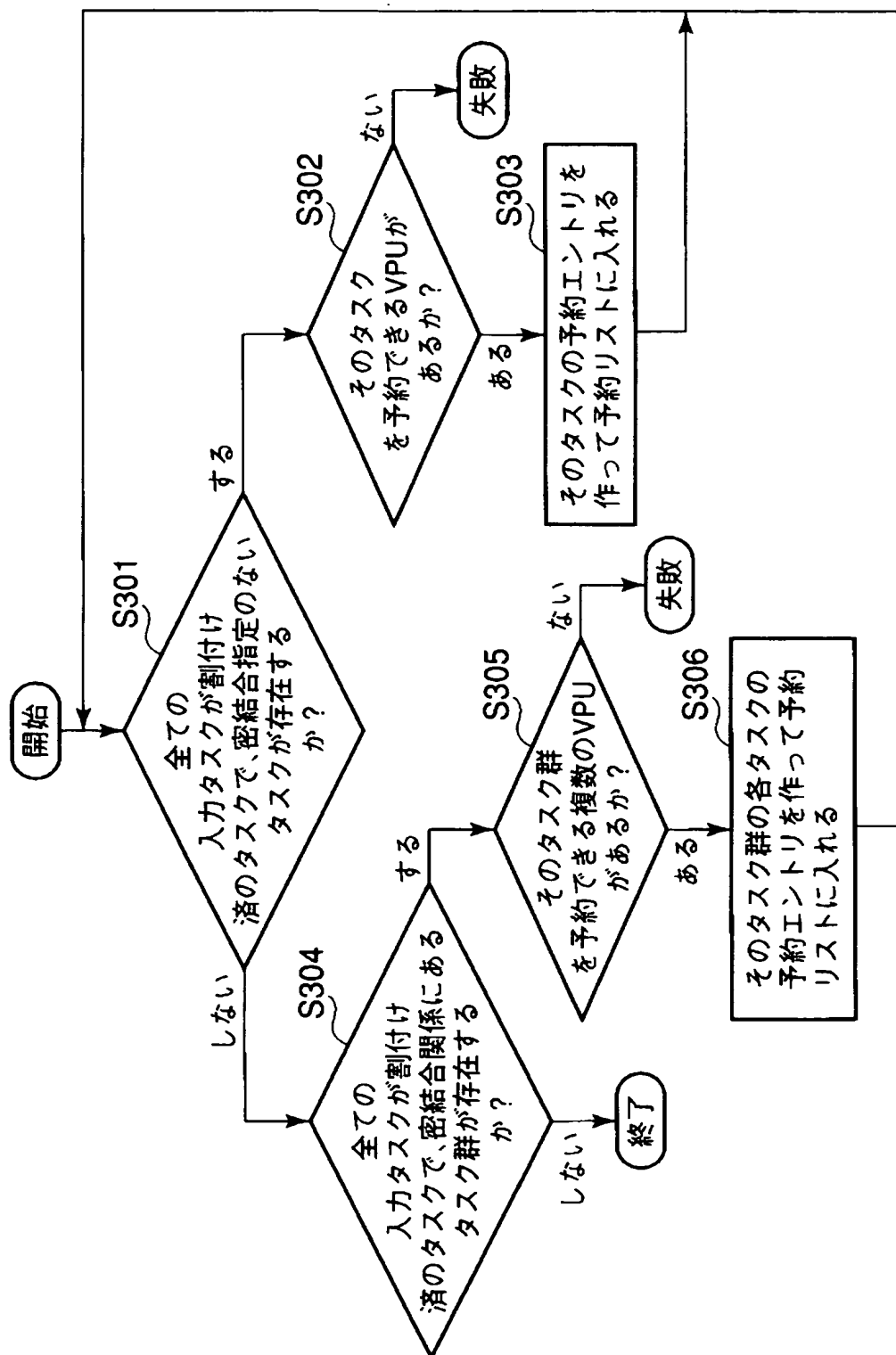
【図 7 2】



【図 73】



【図 74】



【書類名】 要約書

【要約】

【課題】 互いに協調して動作するスレッド間のデータの受け渡しを効率よく実行する。

【解決手段】 リアルタイム処理システムは、密結合スレッドグループというスレッドグループの属性を利用し、スレッドグループが互いに協調して動作するスレッド群を含む密結合スレッドグループであるか否かを判別する。スレッドグループが密結合スレッドグループであることが判別された場合、リアルタイム処理システムは、密結合スレッドグループに属するスレッド群がそれぞれ別のプロセッサ（VPU）によって同時に実行されるように、密結合スレッドグループに属するスレッド群を当該スレッド群の個数分のプロセッサ（VPU）にそれぞれディスパッチするためのスケジューリング処理を実行する。

【選択図】 図 25



特願 2 0 0 3 - 1 8 4 9 7 5

出 願 人 履 歷 情 報

識別番号

[ 0 0 0 0 0 3 0 7 8 ]

1. 変更年月日

2 0 0 1 年 7 月 2 日

[変更理由]

住所変更

住 所

東京都港区芝浦一丁目 1 番 1 号

氏 名

株式会社東芝